WSG Forschungsberichte

WSG-RR 14/98

Ein Softwarepaket zur Lösung von statisch - diskreten Standortallokationsproblemen

[Diplomarbeit]

Gerald Hiebel

Institut für Wirtschafts- und Sozialgeographie

Wirtschaftsuniversität Wien

Department of Economic and Social Geography

Vienna University of Economics and Business Administration

WSG Research Reports

Abteilung für Theoretische und Angewandte Wirtschafts- und Sozialgeographie
Institut für Wirtschafts- und Sozialgeographie
Wirtschaftsuniversität Wien

Vorstand: o.Univ.-Prof. Dr. Manfred M. Fischer
A - 1090 Wien, Augasse 2-6, Tel. ++43-(0)1-31336-4836

Redaktion: Univ.-Ass. Dr. Petra Staufer

WSG-RR 14/98

Ein Softwarepaket zur Lösung von statisch-diskreten Standortallokationsproblemen

[Diplomarbeit]

Gerald Hiebel

WSG-Research Report 14

May 1998

Gedruckt mit Unterstützung
des Bundesministerium
für Wissenschaft und Forschung
in Wien

WSG-Research Reports are presenting complete research outcomes which focus on theoretical, methodical, and empirical scientific problems according to the fields of research at the Department of Economic and Social Geography, Vienna University of Economics and Business Administration.

ISBN 3 85037 077 1

Inhaltsverzeichnis

Abbildungenen und Tabellen

Abbildungen

Tabellen

Vorwort

Diese Diplomarbeit entstand aus der Idee, ein Softwarepaket zur Lösung von statisch-diskreten Standortallokationsproblemen zu entwickeln. Um auch in dieser Forschungsrichtung die Möglichkeiten der modernen Computertechnologie einzusetzen, wurde am Institut für Wirtschafts- und Sozialgeographie unter der Leitung von o. Univ. Prof. Dr. Manfred M. FISCHER diese Aufgabenstellung als Diplomarbeit ausgeschrieben. Dieses Thema entsprach meinem Interesse am Einsatz innovativer EDV-gestützter Methoden zur Lösung räumlicher Problemstellungen. Die Auseinandersetzung mit der Thematik und meine Ausbildung am Institut für Wirtschafts- und Sozialgeographie der WU-Wien und am Institut für Stadt- und Regionalforschung der Österreichischen Akademie der Wissenschaften lenkten mein Forschungsinteresse auf die Verbindung des zu entwickelnden Softwarepaketes mit Geographischen Informationssystemen, kurz GIS. Aus diesem Ansatz entwickelte sich die vorliegende Diplomarbeit.

Mein Dank gilt meinem Institutsvorstand o. Univ. Prof. Dr. Manfred M. FISCHER, der mir im allgemeinen durch seine Forschungsschwerpunkte und die damit verbundenen Lehrveranstaltungen eine anwendungsorientierte Ausbildung im Rahmen meines Studiums ermöglicht hat, und der mich im besonderen durch dieses Diplomarbeitsthema zur Arbeit mit zukunfts- und praxisorientierter Technologie geführt hat. Ohne die ideelle und praktische Unterstützung von Frau Dr. Petra STAUFER, der betreuenden Assistentin, wäre diese Arbeit wohl nicht möglich gewesen. Sie hatte für meine Probleme, ob EDV-spezifischer oder inhaltlicher Natur immer ein offenes Ohr und stand mir mit Rat und Tat zur Seite. Für ihre Geduld und Ausdauer, die vorhandene Arbeit letztendlich in eine entsprechende und verständliche Form und Ordnung zu bringen, bin ich besonders dankbar. Die in dieser Arbeit verwendeten GIS-Datenbestände wurden mir von der WIGeo-GIS GmbH zur Verfügung gestellt.

Insbesonders möchte ich meiner Mutter danken, die durch ihre Offenheit und Großzügigkeit dieses Studium erst ermöglicht hat. Vor allem ist zu erwähnen, daß sie sich auch durch die etwas längere Studiendauer nicht von ihrer Unterstützung abbringen ließ.

Wien, im April 1998 Gerald Hiebel

1. Einleitung

Im Zuge der immer größeren Ausdifferenzierung unserer Gesellschaft und Wirtschaft wird es notwendig, räumliche Strukturen möglichst effizient zu gestalten. Ein wichtiger Teilbereich ist die Planung von **Standorten und Einzugsbereichen**. Besonders bei den Einrichtungen der haushaltsnahen Infrastruktur sind sehr viele Menschen betroffen, schon kleine Verbesserungen in der Standortwahl können insgesamt große Kostenersparnisse oder Zeitgewinne bedeuten. Beispiele sind öffentliche Einrichtungen von zentraler Bedeutung wie Magistrate, Krankenhäuser, Schulen, Postämter, Polizei-, Feuerwehr-, Unfallstationen oder Altersheime. Bei den genannten (Notfall) Einrichtungen liegt das entscheidende Kriterium wiederum darin, daß ein gewisser Zeitfaktor nicht überschritten wird, innerhalb dessen ein Nachfrager versorgt werden kann. Die Planung dieser Einrichtungen wird auch **Standort-Allokation** genannt. Um möglichst effiziente Lösungen zu erhalten, können **räumliche Optimierungsmodelle** herangezogen werden (vgl. FISCHER 1992). Es existieren verschiedene Klassen von räumlichen Optimierungsmodellen, die abhängig von der Problemstellung eingesetzt werden können. Für bestimmte Klassen und Problemstellungen sind Verfahren entwickelt worden, um zu optimalen oder zumindest suboptimalen Lösungen zu gelangen.

Eines der klassischen Standortallokationsprobleme und Ausgangspunkt für Forschungen ist das **P-Median Problem** (vgl. REVELLE und SWAIN 1970), zu dessen Lösung die Klasse der **statisch-diskreten Standortallokationsmodelle** entwickelt wurde. Im Prinzip lassen sich zwei verschiedene Ansätze zur Lösung dieser Problemstellung unterscheiden: **Optimierungsverfahren** (vgl. NARULA 1979, zit. nach. CHURCH 1996), die aber den Nachteil haben, daß sie bei komplexeren Problemstellungen auf Grund des benötigten Rechenaufwandes nicht mehr praktisch durchführbar sind, und **heuristische Verfahren** (vgl. MARANZANA 1964 oder TEITZ und BART 1968, zit. nach. CHURCH 1996), die diese Einschränkung nicht aufweisen, aber auch nicht notwendigerweise zu einer optimalen Lösung führen. Sowohl optimale, als auch heuristische Verfahren zur Lösung von P-Median Problemen wurden in Computer-Programme implementiert.

Ein besonderes Anliegen der vorliegenden Arbeit ist es, die GIS-Technologie nutzbringend mit der Technologie statisch-diskreter Standortallokationsmodelle im Kontext realweltlicher Problemstellungen zu verbinden (vgl. LONGLEY 1996). Dadurch gelingt es, die räumliche Ausgangssituation wie auch die Modellergebnisse nach dem neuesten Stand der Technik zu visualisieren. Dies ist nicht hoch genug zu bewerten, da komplexe räumliche Problemstellungen durch die graphische Darstellung leichter erfaßbar werden. Weiters können **GIS-Datenbestände** als Eingangsdaten für die Modellrechnung herangezogen werden, was eine langwierige und oft kostenintensive Eingabe erspart. Ein Beispiel für eine derartige Verbindung existiert im ARC/INFO NETWORK™ Modul (vgl. ESRI 1990). Nachteil dieses Systems ist vor allem die Komplexität des Softwarepaketes ARC/INFO™, für dessen Benutzung umfangreiches softwarespezifisches GIS Know-How erforderlich ist. Weiters sind die zur Verfügung stehenden Modelltypen sehr eingeschränkt und entsprechen nicht dem neuesten Stand der Modelltechnologie.

Um **räumliche Optimierungsmodelle in der universitären Lehre und in der Praxis anwendbar** zu machen, ist es notwendig, ein Softwarepaket zu entwickeln, das die folgenden Eigenschaften besitzt (vgl. DENSHAM 1996):

- die Möglichkeit, externe Datenbestände (insbesondere GIS-Daten) verwenden zu können,
- die Visualisierung der Problemstellung und der Modellergebnisse,
- eine graphische Möglichkeit zur Spezifikation von Randbedingungen des SDSAM,
- die Verfügbarkeit von analytischen Basisfunktionen eines Desktop-GIS,
- Transparenz des verwendeten Algorithmus,
- Benutzerfreundlichkeit, und
- Offenheit des Systems.

Die **Zielsetzung** dieser Diplomarbeit besteht darin, ein solches **Softwarepaket zu entwickeln**, ein Softwarepaket, das den Einsatz räumlicher Optimierungsmodelle in der Praxis wesentlich erleichtern soll. Im Mittelpunkt steht die Lösung des Standortallokationsproblems. Das angewendete Verfahren geht von einem an der Universität Bremen entwickelten Fortran Programm aus, das in BAHRENBERG (1979)

anhand eines hypothetischen Fallbeispieles beschrieben ist. Hier wird ein heuristisches Verfahren zur Lösung statisch-diskreter Standortallokationsprobleme angewendet, der es ermöglicht, Kapazitätsrestriktionen für die zu bildenden Einzugsbereiche mitzuberücksichtigen.

Diese Arbeit gliedert sich in drei Kapitel und einen Anhang. Das zweite Kapitel behandelt die **methodischen Grundlagen** von räumlichen Optimierungsmodellen. Die wichtigsten Typen der Klasse der statisch-diskreten Standortallokationsmodelle werden definiert und formal beschrieben. Dieses Kapitel lehnt sich an FISCHER (1992) an. Im Mittelpunkt des dritten Kapitels, steht die **Entwicklung** des **Softwarepaketes `LOC-ALLOC`**. Zunächst wird ein **Anforderungsprofil** an das Paket erstellt, das das Softwarepaket erfüllen soll. Anschließend wird das verwendete **heuristische Verfahren** anhand von Struktogrammen und Abbildungen beschrieben. Das Konzept zur Koppelung des Modellsystems (`LOC-ALLOC`) mit dem Desktop-GIS ArcView™ 3.0, sowie das Modellsystem selbst und die graphische Benutzerschnittstelle (`GUI-LOC-ALLOC`) werden in einem Subkapitel zum **softwaretechnischen Ansatz** erklärt.

Das vierte Kapitel gibt eine Einführung in die Arbeit mit `LOC-ALLOC` und der graphischen Benutzerschnittstelle `GUI-LOC-ALLOC`. Dies geschieht zuerst über eine Bedienungsanleitung anhand von zwei **Übungsbeispielen**. Mit ihnen wird der Einstieg in die Arbeit mit `LOC-ALLOC` so einfach wie möglich gestaltet. Weiters soll auch gezeigt werden, wie mit sehr wenigen Eingaben, Demonstrationsbeispiele gelöst werden können. Der zweite Abschnitt dieses Kapitels soll ein realweltliches Anwendungsbeispiel betrachtet und die Lösung beschrieben werden. Das realweltliche Beispiel bezieht sich auf eine Neueinteilung der Postamtsbezirken des südwestlichen Niederösterreichs, etwa um entsprechende Kosten im Kontext der Rationalisierung der Post einsparen zu können. In einem abschließenden Kapitel werden die Ergebnisse dieser Arbeit zusammengefaßt.

Der **Anhang** beinhaltet zuerst die **Installations**- und **Bedienungsanleitung**. Danach folgt die **softwaretechnische Realisierung** in Form des Programmcodes von C++ Programm (`LOC-ALLOC`) und graphischer Benutzerschnittstelle (`GUI-LOC-ALLOC`), die in Avenue (Programmiersprache von ArcView™) programmiert wurde.

2. Methodische Grundlagen

Standort-Allokationsprobleme wie sie beispielsweise bei der Planung von Einrichtungen der haushaltsnahen Infrastruktur auftreten, können mit Hilfe von **räumlichen Optimierungsmodellen** gelöst werden. Aus der Familie der räumlichen Optimierungsmodelle wird im Rahmen dieser Arbeit im weiteren lediglich auf die Klasse **statisch-diskreter Standortallokationsmodelle** näher eingegangen. Ihre formale Beschreibung basiert auf der Graphentheorie. Die **Notationen und Elemente** der formalen Beschreibung werden im weiteren zur Definition von **Erreichbarkeitsmaßen**, **Randbedingungen** und **Zielfunktionen** herangezogen. Unterschiedliche Kombinationen von Randbedingungen und Zielfunktionen liefern verschiedene Typen **elementarer Mitglieder** der Klasse statisch-diskreter Standortallokationsmodelle.

2.1 Klassifikation räumlicher Optimierungsmodelle

Räumliche Optimierungsmodelle lassen sich nach FISCHER (1992) anhand folgender Kriterien klassifizieren (siehe auch Abb. 2.1):

- **Planungszeitraum**

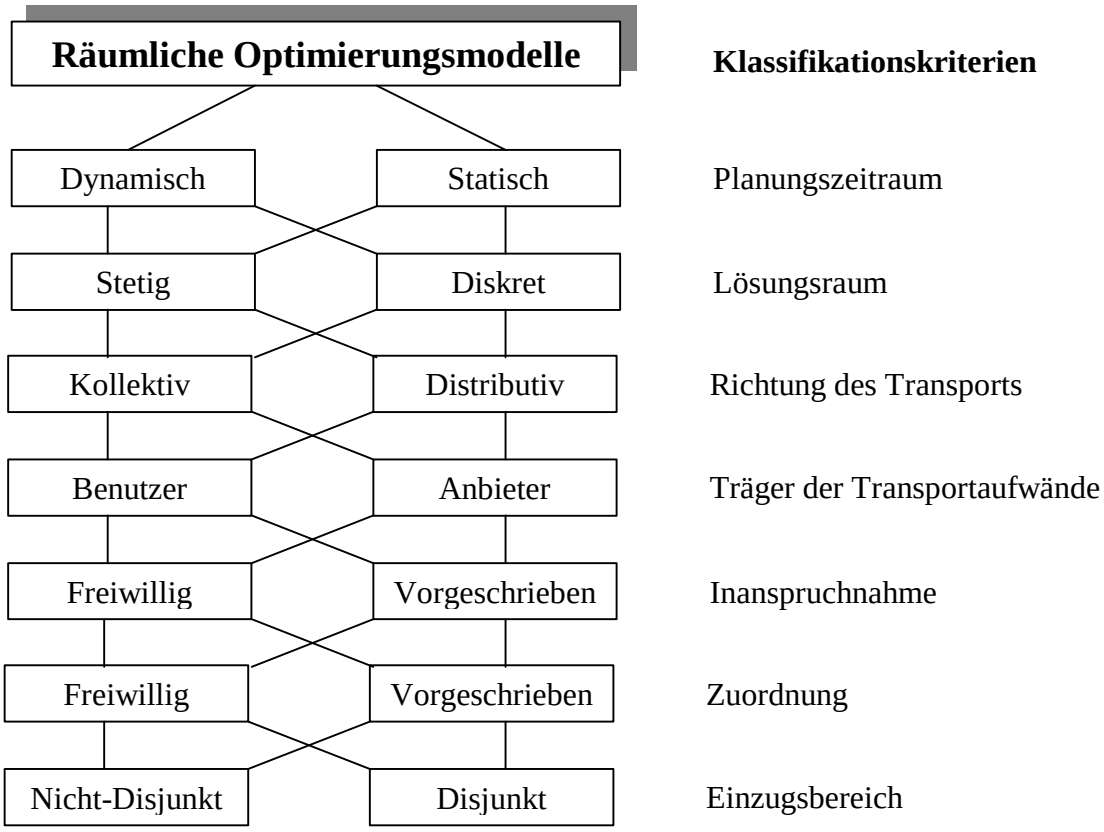
  Ist die Planung auf *einen bestimmten Zeitpunkt* t_0 bezogen, sind *statische* Optimierungsmodelle zur Bestimmung eines Standort-Zuordnungssystems heranzuziehen. Umfaßt der Planungszeitraum hingegen *mehrere Zeitpunkte*, sollten *dynamische* Optimierungsmodelle verwendet werden. Dynamische Optimierungsmodelle lassen sich zwar formal leicht beschreiben, werden aber bei annähernd realistischen Planungsproblemen leicht so umfangreich, daß sie kaum eindeutig lösbar sind. In zahlreichen Fällen können diese Probleme durch die Anwendung statischer Modelle befriedigend gelöst werden.

- **Lösungsraum**

  Wenn prinzipiell *alle Punkte* eines Untersuchungsgebietes als Standorte des Angebots in Frage kommen, handelt es sich um ein *stetiges Problem*. Des weiteren ist bei Planungsproblemen die Anzahl der potentiellen Angebotsstandorte im allgemeinen *endlich* und man spricht daher von einem *diskreten Lösungsraum*.

Die beiden Merkmalsdimensionen Planungszeitraum und Lösungsraum sind von zentraler Bedeutung, da sich die entsprechenden Modelltypen, hinsichtlich der Komplexität der formalen Struktur, der Lösungsmöglichkeiten, wie auch der Anwendungsmöglichkeiten beträchtlich unterscheiden.

Abb. 2.1: Eine Klassifikation räumlicher Optimierungsmodelle (FISCHER, 1992, S. 3)



- **Richtung des Transports**

  Standort-Zuordnungssysteme werden *kollektiv* genannt, wenn die Benutzer überwiegend selbst zu den Angebotsstandorten transportiert werden, oder sich dorthin bewegen müssen (z.B.: Schulen, Kindergärten, Altersheime, Krankenhäuser, Notrufsäulen, etc.). Bei den *distributiven* Systemen werden Güter und Leistungen von den Angebotsstandorten zu dem Benutzer durch entsprechende Transporte gebracht. Beispiele hierfür sind Polizei-, Feuerwehrstationen, medizinische Noteinrichtungen oder Postzustelldienste.

- **Träger der Transportaufwände**

  Bei *kollektiven* Systemen kommen die Benutzer in der Regel *direkt* für ihre eigenen Kosten im weitesten Sinne des Transports auf. Bei *distributiven* Systemen erfolgt die finanzielle Belastung nur *indirekt* und/oder in pauschalierter, distanzunabhängiger Form (Steuern, Gebühren). Bei distributiven Systemen geht es vor allem um Effizienzkriterien, während bei den kollektiven Chancengleichheit und soziale Gerechtigkeit im Vordergrund stehen.

- **Inanspruchnahme der öffentlichen Einrichtungen**

  Zwangsweise Inanspruchnahme öffentlicher Einrichtungen kann entweder aus entsprechenden Gesetzen resultieren (Grundschule, Verwaltungseinrichtungen) oder sich de facto ergeben (Notfalleinrichtungen). Beispiele für Einrichtungen mit *freiwilliger* Inanspruchnahme sind weiterführende Schulen, Universitäten, Einrichtungen der sozialen Infrastruktur, Schwimmbäder, etc. Bei diesen Einrichtungen klaffen Bedarf und Nachfrage möglicherweise weiter auseinander, die zukünftige Nachfrage und damit die Größe der Einrichtungen sind schwieriger zu schätzen. Die Nachfrage ist oft transportaufwandselastisch. Bei der Standortwahl werden daher Punkte mit großem Bevölkerungspotential bevorzugt, mit der Konsequenz einer großen Benachteiligung für einzelne potentielle Nutzer.

- **Zuordnung**

  Ebenso wie die Inanspruchnahme kann die Zuordnung *freiwillig* oder *vorgeschrieben* sein. Bei distributiven Systemen sind die Einzugs- bzw. Absatzgebiete im allgemeinen festgelegt (auch wenn die Inanspruchnahme freiwillig ist).

- **Einzugsbereiche**

  Von wenigen Ausnahmen abgesehen sind *vorgeschriebene (freiwillige*) Zuordnungen mit *disjunkten (nicht-disjunkten*) Einzugsbereichen verbunden.

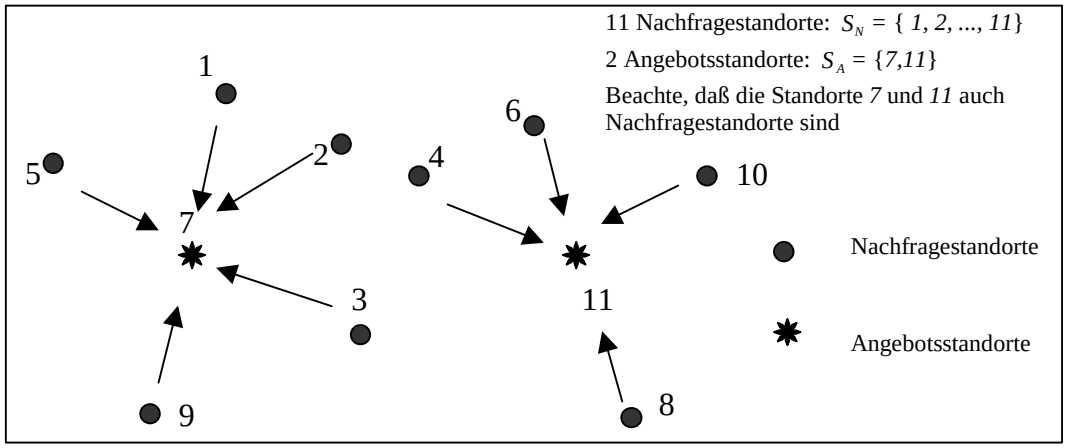
2.2 Das allgemeine statisch-diskrete Standortallokationsmodell (SDSAM) oder die Klasse statisch-diskreter Standortallokationsmodelle

Bei *statischen* Standort-Allokationsproblemen werden die zu bestimmenden Standorte und ihre Zuordnungen für einen längeren Zeitraum als unveränderlich angesehen. *Diskrete* Probleme sind solche, wo nur endlich viele Punkte als potentielle Angebotsstandorte in Frage kommen. Beispiele für diese Fragestellungen sind Altenheime, Schulen, Spielplätze, Kindergärten, Postämter, Polizei-, Feuerwehr-, Unfallstationen, etc. also öffentliche Einrichtungen mit zentraler Bedeutung. Allgemein kann ein statisch-diskretes Standort-Allokationsproblem folgendermaßen formuliert werden:

Gegeben sei eine Menge $S_N = \{i;\ i=1, ..., n\}$ von Standorten, an denen die Bevölkerung konzentriert ist. Diese Punkte (diskreter Lösungsraum) können auch Gebiete repräsentieren an denen die Bevölkerung konzentriert ist (z.B.: Baublöcke, Gemeinden oder andere Verwaltungseinheiten). Sie werden als Verbraucher-, Bevölkerungs- oder Nachfragestandorte bezeichnet.

Gesucht ist eine Menge S_A (S_A Í S_N) von sogenannten Angebotsstandorten und/oder die Zuordnungen der Nachfragestandorte zu den Angebotsstandorten, so daß bestimmte Planungsziele erfüllt werden, wie beispielhaft in Abb. 2.2 für 11 Nachfragestandorte illustriert wird.

Abb. 2.2: Ein hypothetisches Beispiel von Standorten und Zuordnungen



Durch die Zuordnungen zwischen Nachfrage- und Angebotsstandorten werden die *Einzugsbereiche* bzw. *Absatzgebiete* der Angebotsstandorte festgelegt. Sind die Angebotsstandorte gegeben, die Zuordnungen gesucht, so spricht man von einem **Zuordnungs**- oder **Allokationsproblem**. Sind die Nachfragestandorte gegeben, die Angebotsstandorte gesucht, so spricht man von einem **Standort-** oder **Lokalisationsproblem**. Werden die Angebotsstandorte und die Zuordnungen gesucht, so handelt es sich um ein **Standort- Zuordnungs-** oder **Lokalisations- Allokationsproblem.**

Ein **SDSAM** hat grundsätzlich folgende Struktur: Es besteht aus einer Reihe von **Rand- oder Nebenbedingungen** R_g ($g; g=1, ..., G$), unter denen eine **Zielfunktion** Z_h ($h; h=1, ..., H$) zu maximieren oder zu minimieren ist. Im Kontext von Standort-Zuordnungsproblemen entsprechen die Randbedingungen den Planungszielen die in jedem Fall erreicht werden müssen. Die Zielfunktion dient dazu, bestimmte Eigenschaften des Standort-Zuordnungssystems (z.B. die Erreichbarkeit bzw. die Anzahl der Einrichtungen, etc.) zu optimieren. Dabei handelt es sich um ein möglichst gut zu erreichendes Planungsziel. Insofern sind die in den Randbedingungen repräsentierten Planungsziele von primärer, übergeordneter Bedeutung, die Zielfunktion entspricht einem minder wichtigem Planungsziel. Formal betrachtet hat die Einbeziehung von Zielfunktionen und deren Maximierung bzw. Minimierung vor allem den Sinn, eine eindeutige Lösung für ein Standort-Allokationsproblemproblem zu erzielen für den Fall, daß es für die Randbedingungen mehrere zulässige Lösungen gibt**.** *Lineare Optimierungsmodelle* sind solche, in denen sowohl die Randbedingungen wie auch die Zielfunktion linear sind.

Graphentheoretische Grundlagen zur Modellierung der Modellklasse

Die Modellierung der SDSAM basiert auf der **Graphentheorie**. Ein **Graph** $G=(V,E)$ besteht aus zwei endlichen Mengen,

- V, eine nichtleere Menge von **Knoten**, und
- E, eine Menge von **Kanten**,

so daß jede Kante $e \in E$ einem ungeordneten Paar von Knoten (u,v) mit $u, v \in V$ zugeordnet ist, die als Endknoten von e bezeichnet werden.

Zur Verdeutlichung betrachten wir ein Standort-Zuordnungssystem, in dem die Knoten Ortschaften und die Kanten des Graphen Straßenverbindungen zwischen den Ortschaften repräsentieren. Für gewisse Standort-Allokationsprobleme (z.B. innerhalb von städtischen Gebieten, wo es Einbahnregelungen oder Abbiegevorschriften gibt) kann es notwendig sein, die Kanten eines Graphen mit einer Richtung zu versehen. Jede Kante e ist dann einem geordneten Paar von Knoten (u, v) zugeordnet. In diesem Fall spricht man von einem **gerichteten Graphen.** In der graphischen Darstellung wird die Richtung einer Kante durch einen Pfeil gekennzeichnet.

Notation und Elemente zur formalen Beschreibung und Bewertung von Standort-Zuordnungssystemen

$S_N = \{i; i=1, ..., n\}$	Menge der Nachfragestandorte (entspricht den **Knoten** des Graphen), auch Verbraucher- oder Bevölkerungsstandorte genannt, wobei n die Anzahl der Nachfragestandorte angibt,
b_i $(i=1, ..., n)$	Bedarf des Nachfragestandortes i nach dem zentralen Gut oder der Leistung,
$S_P = \{j; j=1, ..., n\}$	Menge der potentiellen Angebotsstandorte,
S_1	Menge der exogen vorgegeben Angebotstandorte ($S_1 \subseteq S_P$),
$S_A = \{j; j=1, ..., m, m \leq n\}$	Menge der Angebotsstandorte ($S_1 \subseteq S_A \subseteq S_P \subseteq S_N$), wobei m die Anzahl der Angebotsstandorte angibt,
S_0	Menge der als Angebotstandorte ausgeschlossenen Nachfragestandorte ($S_0 \cap S_P = \emptyset$),

$b = \sum_{i \in S_N} b_i$ Gesamtnachfrage („Gesamtbedarf“) der zu versorgenden Fläche/Bevölkerung,

d_{ij} verallgemeinerte Reisekosten (Transportaufwand) von Standort i zum Standort j (Transportaufwand in einfachster Form gemessen als geographische Distanz),

C_0, C_1, a_1 exogen festzulegende Parameter; C_0 ist ein maximaler Transportaufwand (kein Nachfragestandort i darf zu seinem Angebotsstandort j einen größeren Transportaufwand als C_0 Transportkosteneinheiten haben); C_1 ist eine andere Transportaufwandsobergrenze und a_1 eine Obergrenze für den Bedarf (höchstens ein bestimmter Anteil a_1 der Bevölkerung [des Bedarfs] darf weiter als eine gegebene Obergrenze C_1 von dem zugeordneten Angebotsstandort entfernt sein),

$x_{ij} = \begin{cases} 1 \text{ falls } j \text{ Angebotsstandort ist und der Nachfragestandort } i \\ \quad \text{dem Angebotsstandort } j \text{ zugeordnet wird} \\ 0 \text{ sonst} \end{cases}$

$\sum_{i \in S_N} x_{ii}$ Die Anzahl der Nachfragestandorte mit wenigstens einem Vorkommen der Einrichtung,

$\sum_{i \in S_N} b_i x_{ij}$ Größe der Einrichtung am Standort j gemessen als Nachfragepotential,

k exogen festzulegender Parameter, der die Gesamtzahl der Zweit-, Dritt-, Viert-, ... -Einrichtungen angibt,

$k + \sum_{i \in S_N} x_{ii}$ absolute Häufigkeit der Einrichtungen,

M exogen festzulegender Parameter, der die Höchstgrenze der absoluten Häufigkeit der Einrichtungen angibt,

K_U exogen festzulegender Parameter, der eine Kapazitätsuntergrenze der Einrichtung(en) angibt,

K_O exogen festzulegender Parameter, der eine Kapazitätsobergrenze der Einrichtung(en) angibt,

G exogen festzulegender Parameter; G ist ein minimaler Transportaufwand; erst ab einem größeren Transportaufwand als G Transportkosteneinheiten vom Nachfragestandort i zum Angebotsstandort j geht der Bedarf b_i des Nachfragestandortes i in die Zielfunktion Z_3 ein,

a exogen zu wählender Gewichtungskoeffizient,

b exogen zu wählender Gewichtungskoeffizient.

Durch die **Zuordnungsmatrix** (x_{ij}) werden die Angebotsstandorte und die Zuordnungen festgelegt. Für das Beispiel in Abb. 2.2 ergibt sich die in Abb. 2.3 dargestellte Zuordnungsmatrix. Formal betrachtet, besteht das Standort-Zuordnungsproblem darin, die Werte der Variablen x_{ij} entsprechend den Planungszielen festzulegen. Die Angebotsstandorte i sind nämlich genau diejenigen Nachfragestandorte i, für die $x_{ii} = 1$ gilt. Die Anzahl der Angebotsstandorte mit wenigstens einer Einrichtung ist somit $\sum_{i \in S_N} x_{ii}$, und der Bedarf am Nachfragestandort, der durch den Angebotsstandort j gedeckt wird, beträgt $b_i x_{ij}$. Für die Größe der Einrichtungen am Standort j gemessen als Nachfragepotential ergibt sich: $\sum_{i \in S_N} b_i x_{ij}$.

Abb. 2.3: Matrix für Standorte und Zuordnungen für das Beispiel in Abb. 2.2

		Angebotsstandorte										
	i \ j	1	2	3	4	5	6	7	8	9	10	11
Nachfragestandorte	1	0	0	0	0	0	0	**1**	0	0	0	0
	2	0	0	0	0	0	0	**1**	0	0	0	0
	3	0	0	0	0	0	0	**1**	0	0	0	0
	4	0	0	0	0	0	0	0	0	0	0	**1**
	5	0	0	0	0	0	0	**1**	0	0	0	0
	6	0	0	0	0	0	0	0	0	0	0	**1**
	7	0	0	0	0	0	0	**1**	0	0	0	0
	8	0	0	0	0	0	0	0	0	0	0	**1**
	9	0	0	0	0	0	0	**1**	0	0	0	0
	10	0	0	0	0	0	0	0	0	0	0	**1**
	11	0	0	0	0	0	0	0	0	0	0	**1**

Erreichbarkeitsmaße

Die im Rahmen dieser Arbeit betrachtete Klasse statisch-diskreter Standortallokationsmodelle ist durch eine Reihe von Erreichbarkeitsmaßen, Randbedingungen und Zielfunktionen definiert, die optional gewählt werden können und so zu spezifizierten Modellen der Modellklasse führen. Randbedingungen und Zielfunktionen basieren teilweise auf den Erreichbarkeitsmaßen. Wir unterscheiden drei verschiedene Erreichbarkeitsmaße, die zur Modellspezifikation herangezogen werden können (vgl. FISCHER, 1992):

(M_1) $\quad M_1(a,C) = \sum_{\substack{i,j\in S_N \\ d_{ij}\geq C_0}} b_i d_{ij}^a x_{ij}$ $\quad (a > 0,\ C_0$ Parameter)

(M_2) $\quad M_2(C_1) = \sum_{\substack{i,j\in S_N \\ d_{ij}\geq C_1}} b_i x_{ij}$ $\quad (C_1$ Parameter)

(M_3) $\quad M_3 = \max_{i,j\in S_N} d_{ij} x_{ij}$

M_1 ($a=1$) ist das gängigste Erreichbarkeitsmaß unter der Annahme, daß die Versorgung der Bevölkerung jeweils durch Einzelfahrten zwischen Angebots- und Nachfragestandorten erfolgt; das heißt verkettete Fahrtrouten und Mehrzweckfahrten werden nicht in Betracht gezogen. Dieses Erreichbarkeitsmaß berücksichtigt allerdings die weit entfernten Nachfragestandorte zu wenig (daher $a=1$) und ist vor allem im Fall von Notfalleinrichtungen ebenso wie bei kollektiven Systemen weniger geeignet. Je größer a ist, desto stärker werden die extremen Distanzen gewichtet. Das Maß M_2 ist ein Spezialfall von M_1, den man erhält, wenn $a=0$ und $C_0 = C_1$. M_3 ist dasjenige Erreichbarkeitsmaß, das die extremen Distanzen am stärksten gewichtet.

Die **monetären Kosten** eines Standort-Zuordnungssystems werden ohne Transportaufwand betrachtet, die bereits in die Erreichbarkeitsmaße eingehen. Sie werden in erster Näherung als proportional zur Anzahl der Einrichtungen angenommen. Stellt man die fixen plus die variablen Kosten der Nachfrage gegenüber, so erhält man ein Maß für die Effizienz eines Standort- Zuordnungssystems.

Randbedingungen (Constraints)

Ein SDSAM setzt sich aus **Randbedingungen** und einer **Zielfunktion** zusammen, die nun mit den oben genannten Definitionen und Maßen zur Beschreibung von Standort-Zuordnungssystemen formuliert werden sollen. Bei den Randbedingungen handelt es sich um übergeordnete Planungsziele, die es in jedem Fall zu erfüllen gilt. Im Rahmen der vorliegenden Arbeit werden folgend Randbedingungen betrachtet:

(R_1) **Vollständige Versorgung**

An erster Stelle ist sicherzustellen, daß alle Nachfragestandorte versorgt sind. Jeder Nachfragestandort muß mindestens und, wegen der Voraussetzung disjunkter Einzugsbereiche, genau einem Angebotsstandort zugeordnet werden.

Für alle $i \in S_N$ gilt $\sum_{j \in S_N} x_{ij} = 1$.

(R_2) Kompaktheit der Einzugsbereiche

Die Einzugsbereiche sollen eine gewisse Kompaktheit aufweisen:

Für alle $i, j \in S_N$ mit $x_{ij} = 1$ gibt es p Punkte $i = i_1, ..., i_p = j$, so daß für k=1, ..., p-1 gilt i_k ist benachbart zu i_{k+1} und $x_{i_k j} = 1$.

Diese Bedingung hängt in ihrer konkreten Ausformung davon ab, wie "benachbart“ definiert ist. Sind die Punkte i Knoten in einem Verkehrsnetz, so können zwei Punkte z.B. als benachbart gelten, wenn sie durch eine Kante direkt miteinander verbunden sind. Werden Flächen durch i repräsentiert, kann man zwei Punkte dann als benachbart bezeichnen, wenn ihre Flächen eine gemeinsame Grenze haben (also topologisch zusammenhängend sind). Um mit dieser Randbedingung R_2 arbeiten zu können muß eine Nachbarschafts- oder Konnektivitätsmatrix (Kon_{ij}) vorliegen . Für diese Matrix gilt:

$$Kon_{ij} = \begin{cases} 1 & \text{falls } i \text{ und } j \text{ benachbart sind} \\ 0 & \text{sonst} \end{cases}$$

(R_3) Bevölkerung eines Angebotsstandortes ist diesem auch zugeordnet

Für alle $i, j \in S_N$ gilt: $x_{jj} \geq x_{ij}$.

(R_4) Maximaler Transportaufwand (C_0)

Mindestanforderungen an die **Erreichbarkeit** können je nach dem gewählten Maß in verschiedener Weise dargestellt werden, wobei M_1 hier weniger gut geeignet ist. Dagegen ist bei Einrichtungen für distanzempfindliche Benutzer und bei Notfalleinrichtungen eine Bedingung folgender Art (vgl. M_3) zu stellen:

$$\max_{i, j \in S_N} d_{ij} x_{ij} \leq C_0 .$$

Kein Nachfragestandort i darf zu seinem Angebotsstandort j einen größeren Transportaufwand als C_0 Transportkosteneinheiten haben.

(R_5) **Höchstens ein bestimmter Anteil** a_1 **der Bevölkerung** (des Bedarfs) darf mehr als eine **gegebene Obergrenze** C_1 von dem zugeordneten Angebotsstandort entfernt sein:

$$\left(\sum_{\substack{i,\, j \in S_N \\ d_{ij} \geq C_1}} b_i x_{ij} \Big/ \sum_{i \in S_N} b_i \right) \leq a_1 .$$

Anmerkung: (R_4) ist ein Spezialfall von (R_5) mit $a_1 = 0$ und $C_1 = C_0$.

(R_6) **Höchstgrenzen für die absolute Häufigkeit**

$\sum_{i \in S_N} x_{ii}$ gibt die Anzahl der Nachfragestandorte mit mindestens einer Einrichtung an. Es ist möglich, daß die gleiche Einrichtung an einem Standort mehrmals vorhanden ist oder sein soll. Die Forderung nach einer Höchstgrenze der absoluten Häufigkeit lautet:

$$k + \sum_{i \in S_N} x_{ii} \leq M ,$$

wobei *M* die Anzahl der Einrichtungen bezeichnet und *k* die Gesamtzahl der Zweit-, Dritt-, Viert-, ... Einrichtungen bezeichnet. Ist die Anzahl der Angebotsstandorte fix vorgegeben, so würde die verschärfte (R_6) lauten $k + \sum_{i \in S_N} x_{ii} = M$, wobei *k* und *M* fix vorgegeben wären.

(R_7) **Kapazitätsuntergrenzen für Angebotsstandorte**

Es können auch - aus ökonomischen, sozialen und Gründen der Funktionsfähigkeit - Grenzen für die Größe der Einrichtungen an den potentiellen Angebotsstandorten bzw. der Einzugsbereiche der potentiellen Angebotsstandorte gefordert werden. Die Forderung nach einer Kapazitätsuntergrenze lautet:

Für alle $j \in S_N$ mit $x_{jj} = 1$: $\sum_{i \in S_N} b_i x_{ij} \geq K_U$,

wobei K_U ein Parameter ist der festgelegt werden muß.

(R_8) **Kapazitätsobergrenzen für Angebotsstandorte**

Für alle $j \in S_N$ mit $x_{jj} = 1$: $\sum_{i \in S_N} b_i x_{ij} \leq K_O$,

wobei K_O ein Parameter ist der festgelegt werden muß.

(R_9) **Vorgegebene Angebotsstandorte**

Standort-Zuordnungssysteme sind selten vollständig zu planen. Meist sind bereits einige Angebotsstandorte gegeben, und nur einige neue werden gesucht:

$x_{ii} = 1$ für alle $i \in S_1$ (Menge der exogen vorgegeben Angebotstandorte ($S_1 \subseteq S_P$)).

(R_{10}) **Ausgeschlossene Angebotsstandorte**

Öffentliche Einrichtungen benötigen eine Fläche mit bestimmten Mikroeigenschaften. Um mögliche Makrostandorte mit fehlenden geeigneten Mikroeigenschaften von vornherein als potentielle Angebotsstandorte auszuschließen, wird gefordert:

$x_{ii} = 0$ für alle $i \in S_0$ (Menge der als Angebotstandorte ausgeschlossenen Nachfragestandorte).

Zielfunktionen (Objectives)

Sind die Randbedingungen ($R_1 - R_{10}$), oder der für das zu behandelnde Problem relevante Teil erfüllt, so kann das Standort-Zuordnungssystem hinsichtlich bestimmter Eigenschaften (z.B. der Erreichbarkeit) optimiert werden. Zu diesem Zweck werden in der vorliegenden Arbeit folgende Zielfunktionen optional zur Verfügung gestellt:

(Z_1) **Minimiere die maximale Distanz**

Minimiere $Z_1(x_{ij}) = \max_{i,j \in S_N} d_{ij} x_{ij}$.

(Z_2) **Minimiere den Anteil der Bevölkerung (*b*) die eine höhere Distanz als C_1 zurückzulegen hat**

Minimiere $Z_2\ (x_{ij}) = \left(\sum_{\substack{i,j \in S_N \\ d_{ij} \geq C_1}} b_i x_{ij} \Big/ \sum_{i \in S_N} b_i \right)$.

(Z_3) **Minimiere eine Funktion, die Bedarf und Distanz in Beziehung setzt**

Minimiere $Z_3\ (x_{ij}) = \sum_{\substack{i,j \in S_N \\ d_{ij} \geq G}} b_i^a\, d_{ij}^b\, x_{ij}$.

Diese Funktion ist ein häufig angewandtes Erreichbarkeitsmaß, a und b sind Gewichtungsparameter, mit deren Hilfe man die Bedeutung der beiden Komponenten („Bedarf“ und Distanz) festlegen kann. Die Minimierung dieser Funktion führt im Fall $a=b=1$ und $G=0$ zu sogenannten Medianstandorten, für $a=1$, $b=2$ und $G=0$ zu arithmetischen Mittelzentren. Die Zielfunktionen (Z_1) - (Z_3) minimieren Transportaufwandsmaße bzw. maximieren die in verschiedener Weise definierte Erreichbarkeit (vgl. (M_1) - (M_3)).

(Z_4) Um die **absolute Häufigkeit der Einrichtungen zu minimieren**, was aus Kostengründen sinnvoll erscheinen kann, kann wird die folgende Zielfunktion herangezogen werden:

Minimiere $Z_4\ (x_{ij}) = k + \sum_{i \in S_N} x_{ii}$.

(Z_5) Um die **Leistungen der Einrichtung** so weit wie **möglich lokal zur Verfügung zu stellen**, kann eine Zielfunktion herangezogen werden, die die Zahl der Angebotsstandorte maximiert.

Maximiere $Z_5\ (x_{ij}) = \sum_{i \in S_N} x_{ii}$.

2.3 Elementare Mitglieder der Klasse statisch-diskreter Standortallokationsmodelle

Je nachdem, welche Randbedingungen und welche Zielfunktion für ein konkretes Planungsproblem gewählt werden, sind eine Reihe verschiedener Standort-Allokationsmodelle denkbar. Im folgenden werden einige grundlegende Typen vorgestellt, die in der Literatur besondere Beachtung gefunden haben. Bei den ersten drei Typen handelt es sich um Standort-Zuordnungsmodelle ohne Kapazitätsrestriktionen im Gegensatz zu den Modelltypen 4 und 5, die Kapazitätsrestriktionen beinhalten (vgl. FISCHER 1992). Besonders interessant für konkrete Planungsprobleme sind Modelle, die sich aus der Kombination von mehreren der elementaren Typen 1-5 ergeben.

Modelltyp 1: **Das vollständige Überdeckungsmodell** (Location Set Covering Model)

- Problemstellung: Ein Gebiet ist mit möglichst wenig Angebotsstandorten so zu versorgen, daß die Entfernungen zwischen den Nachfragestandorten und den zugeordneten Angebotsstandorten eine gegebene Grenze nicht überschreiten.
- Randbedingungen: R_1, R_4, R_9, R_{10}
- Zielfunktion: Z_4 mit k=0

Modelltyp 2: Modelle zur **Maximierung der Erreichbarkeit bei gegebener Anzahl der Angebotsstandorte**

- Problemstellung: Ein Gebiet mit Angebotsstandorten so zu versorgen, daß die Erreichbarkeit bei gegebener Anzahl der Angebotsstandorte bzw. absoluter Häufigkeit der Einrichtungen maximiert wird. Bei der Wahl von Z_2 spricht man vom **maximalen Überdeckungsproblem.**
- Randbedingungen: R_1, R_2, R_3, R_6, R_9, R_{10}
- Zielfunktion: Z_1 oder Z_2 oder Z_3

Modelltyp 3: Modelle zur **Maximierung der Erreichbarkeit bei gegebener Anzahl der Angebotsstandorte und zusätzlichen Erreichbarkeitsrestriktionen**

- Problemstellung**:** Modelle vom Typ 2, die um die Randbedingungen R_4 und/oder R_5 erweitert werden. Dadurch wird zusätzlich eine Mindestanforderung an die Erreichbarkeit gestellt, bei deren Erfüllung dann ein weiteres Erreichbarkeitskriterium optimiert wird. Das **P-Median Problem** kann mit diesem Modelltyp gelöst werden.
- Randbedingungen: R_1, R_2, R_3, R_4, R_5, R_6, R_9, R_{10}
- Zielfunktion: Z_1 oder Z_2 oder Z_3

Modelltyp 4: Modelle mit **maximaler Angebotsdispersion**

- Problemstellung: In ländlichen dünn besiedelten Gebieten ist die Mindestgröße zentraler Einrichtungen bzw. ihrer Einzugsbereiche ein kritischer Parameter. Bei geringer Bevölkerungsdichte ist häufig die Frage von Interesse, an wie vielen Standorten eine Einrichtung vertreten sein kann, wenn die Einzugsbereiche eine gewisse Mindestgröße aufweisen müssen und zusätzlich noch minimale Anforderungen an ihre Kompaktheit gestellt werden.
- Randbedingungen: R_1, R_2, R_3, R_7, R_9, R_{10}
- Zielfunktion: Z_5

Modelltyp 5: Modelle mit **Kapazitätsrestriktionen**

- Problemstellung**:** Die Nachfragestandorte sollen *M-k* fest vorgegebenen Angebotsstandorten so zugeordnet werden, daß die Kapazitätsrestriktionen R_7 und R_8 erfüllt sind, wobei die Einzugsgebiete eine gewisse Kompaktheit (R_2 und R_3) aufweisen müssen. Darüberhinaus sollen die Zuordnungen so bestimmt werden, daß die Erreichbarkeit (gemessen an Z_1, Z_2 oder Z_3) möglichst gut wird.
- Randbedingungen : R_1, R_2, R_3, R_6, R_7, R_8, R_9, R_{10}
- Zielfunktion : Z_1 oder Z_2 oder Z_3

3. Entwicklung des Softwarepaketes LOC-ALLOC

Ein Ziel dieser Arbeit besteht darin, ein Softwarepaket (LOC-ALLOC) zu entwickeln, das es ermöglicht, ausgewählte Typen von SDSAM zur Verfügung zu stellen, die in Kapitel 2 spezifiziert wurden. In diesem Kapitel wird zunächst ein **Anforderungsprofil** erstellt, das softwaretechnisch als Pflichtenheft des Programmpaketes verstanden werden kann. Anschließend wird das zur Lösung von diskreten Standortallokationsproblemen als Grundlage herangezogene **heuristische Verfahren** beschrieben, das in einem C++ Programm implementiert wird. Dieses Programm (LOC-ALLOC) ist als ein Modellsystem zu verstehen.

Um das Modellsystem mit einem **GIS zu koppeln**, existieren verschiedene Ansätze (vgl. FISCHER 1996). Eine Möglichkeit ist die vollständige Integration in ein GIS, wie sie im ARC/INFO NETWORK™ Modul realisiert wurde (vgl. ESRI 1990). Eine weitere Möglichkeit ist die lose Koppelung des Modellsystems mit einem GIS. Beide Programmpakete werden unabhängig voneinander verwendet und der Datenaustausch erfolgt beispielsweise über Textdateien. Der für LOC-ALLOC gewählte Ansatz basiert auf dieser Idee. Die benötigten Daten sind in vier Textdateien gespeichert. Basierend auf diesen Textdateien, die Graphenstruktur und Problemspezifikation enthalten, wird die Modellrechnung durchgeführt und das Modellergebnis soll wieder in einer Textdatei abgespeichert werden. Zusätzlich wurde eine graphische Benutzeroberfläche (GUI-LOC-ALLOC) entwickelt, die eine Menüsteuerung von LOC-ALLOC ermöglicht, und die zur Visualisierung von Problemstellung und Modellergebnissen herangezogen werden kann (vgl DENSHAM 1994). GUI-LOC-ALLOC wurde mit AVENUE™, der Programmiersprache des Desktop-GIS ArcView™ 3.0, entwickelt. Dadurch ist es möglich, GUI-LOC-ALLOC in das Desktop-GIS ArcView™ 3.0 zu integrieren, und damit die Analysefunktionen von LOC-ALLOC innerhalb des GIS in einer einheitlichen Benutzeroberfläche zur Verfügung zu stellen. Der Ablauf einer Modellrechnung, sowie das Zusammenspiel der einzelnen Komponenten (LOC-ALLOC, Textdateien, GUI-LOC-ALLOC und ArcView™ 3.0) wird im **softwaretechnischen Ansatz** dargestellt.

3.1 Anforderungsprofil

Vor der Erstellung eines Softwarepaketes ist es notwendig zu spezifizieren, was es leisten soll, um seine Struktur von vornherein auf diese Anforderungen abzustimmen (vgl DENSHAM 1996):

- **Möglichkeit, externe Datenbestände (insbesonders GIS-Daten) verwenden zu können**
  Es muß relativ leicht möglich sein, GIS-Datenbestände so zu modifizieren, daß sie als Datengrundlagen für LOC-ALLOC verwendet werden können. Auch ermöglicht ein GIS mit seinen analytischen Funktionalitäten die Modifikation der Datengrundlagen je nach der Problemstellung, z.B. die Umrechnung der Distanzen zwischen den Standorten von Straßen- in Luftlinienkilometern.

- **Visualisierung der Problemstellung und der Modellergebnisse**
  Existieren für die gegebene Problemstellung GIS-Daten, so soll es innerhalb des Softwarepaketes möglich sein, die Problemstellung graphisch darzustellen, ebenso wie die Mobellergebnisse.

- **Graphische Möglichkeit zur Spezifikation von Randbedingungen des SDSAM**
  Standorte sollen über ihre graphische Darstellung direkt auswählbar sein, um sie beispielsweise als Angebotsstandorte auszuschließen oder fix vorzugeben.

- **Verfügbarkeit von analytischen Basisfunktionen eines Desktop-GIS**
  GIS-Basisfunktionen, wie Distanzmessungen oder Informationsabfragen zu den Standorten sollen verfügbar sein, da sie dem Benutzer eine bessere Informationsmöglichkeit geben und so helfen, den Modellierungsprozeß effizienter zu gestalten.

- **Benutzerfreundlichkeit**
  Benutzerfreundlichkeit des Systems soll durch eine Menüsteuerung erzielt werden.

- **Offenheit des Systems**

  Das Programm, in dem der heuristische Algorithmus implementiert ist, soll so beschaffen sein, daß das Modellsystem leicht um neue Randbedingungen und Zielfunktionen (d.h. Modelltypen) erweiterbar ist. Es ist ferner wünschenswert, die Möglichkeit zu eröffnen, alternative heuristische Algorithmen leicht in das System aufnehmen zu können.

3.2 Beschreibung des heuristischen Verfahrens

Um die Ergebnisse der Standortallokationsberechnungen nachvollziehbar zu machen und ein Verständnis zu bekommen, warum bestimmte Ausgangssituationen zu den gelieferten Ergebnissen führen, ist es wichtig, die Arbeitsweise der Algorithmen zu verstehen. Dies gilt im besonderen für das hier verwendete heuristische Verfahren, welches unter manchen Voraussetzungen suboptimale Lösungen liefert.

Das zur Lösung der spezifizierten Standortallokationsprobleme herangezogene heuristische Verfahren (vgl. BAHRENBERG 1979) ist in seiner Struktur dem von **Maranzana** (1964) ähnlich (vgl. CHURCH 1996). Dieses Verfahren hat den Vorteil, daß Standortallokationsprobleme mit **Kapazitätsrestriktionen** behandelt werden können. Es löst Standortallokationsprobleme mittels der elementaren Modelltypen 2, 3 und 5 oder einer Kombination aus ihnen, mit der Einschränkung, daß die Anzahl der Angebotsstandorte fix vorgegeben sein muß (verschärfte R_6). Auf die Randbedingungen bezogen, kann vereinfacht gesagt werden, daß es sich um eine Kombination der Randbedingungen R_1 - R_5 und R_7 - R_{10} handelt, wobei R_1 und R_3 für alle Modelltypen gefordert werden. Bei den Zielfunktionen kann optional zwischen Z_1, Z_2 oder Z_3 gewählt werden.

Das Verfahren gliedert sich grundsätzlich in **zwei Teilverfahren**, wobei das erste dazu dient, ein Standort-Zuordnungssystem zu finden, das den geforderten Randbedingungen entspricht (ohne Berücksichtigung von R_4 und R_5) . Im zweiten Teilverfahren wird das gefundene Standort-Zuordnungssystem dahingehend optimiert, daß die für das spezifizierte Standortallokationsproblem gewählte Zielfunktion ein Minimum erreicht. Anschließend wird geprüft, ob R_4 und R_5 ebenfalls erfüllt wurden.

Teilverfahren zum Auffinden eines Standort-Zuordnungssystems, das den geforderten Randbedingungen (ohne R_4 und R_5) entspricht

Die spezifizierten Standortallokationsprobleme lassen sich in zwei Teilprobleme, das **Standortproblem** und das **Zuordnungsproblem** gliedern. Gegeben ist eine Menge S_N der Nachfragestandorte und gesucht sind:

1) eine vorgegebene Anzahl von *m* Angebotsstandorten (**Standortproblem**) und
2) die Zuordnungen der Nachfragestandorte zu den Angebotsstandorten, auch als *Einzugsbereiche* der Angebotsstandorte bezeichnet (**Zuordnungsproblem**).

ad 1) Der Ansatz zur Lösung des **Standortproblems** geht von einer Menge S_A beliebig gewählter Angebotsstandorte aus. Aus der Menge der potentiellen Angebotsstandorte S_P wird im Zufallsverfahren eine Menge von *m* Angebotsstandorten gewählt. Hierbei wird berücksichtigt, daß $S_0 \cap S_P = \varnothing$ und $S_1 \subseteq S_P$. Diese im weiteren als *Startkonfiguration* bezeichnete Menge von Angebotsstandorten wird als Ausgangspunkt für die Lösung des Zuordnungsproblems benutzt.

ad 2) Zur Lösung des **Zuordnungsproblems** werden für die in der Startkonfiguration enthaltenen Angebotsstandorte *Einzugsbereiche* gebildet, die den Randbedingungen R_2 - R_5 und R_7 - R_{10} entsprechen und alle Nachfragestandorte beinhalten (R_1). Sind bestimmte Randbedingungen nicht gestellt, werden für die entsprechenden Parameter Voreinstellungen angenommen, so daß diese Randbedingungen bei der Ausführung des Algorithmus nicht berücksichtigt werden.

Voreinstellungen:

(R_2) Jeder Knoten ist zu jedem anderen Knoten benachbart

(R_7) $K_U = 0$

(R_8) $K_O = \infty$

(R_9) $S_1 = \varnothing$

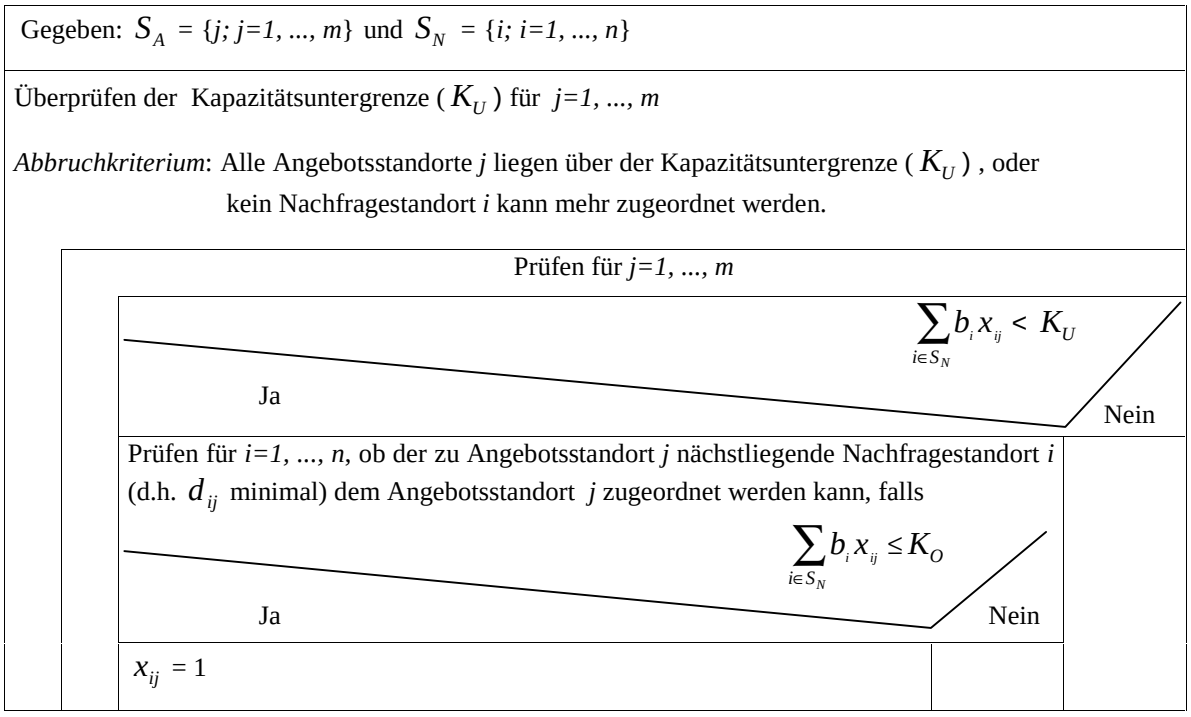
(R_{10}) $S_0 = \varnothing$

Bei der Lösung des Zuordnungsproblems ist zwischen zwei Arten von Standortallokationsproblemen zu unterscheiden:

- **mit unterer Kapazitätsbeschränkung** (R_7),
- **ohne untere Kapazitätsbeschränkung.**

Zur Lösung beider Problemstellungen werden Algorithmen herangezogen, die sich in 2 Stufen gliedern. Handelt es sich um eine **Problemstellung mit unterer Kapazitätsbeschränkung**, so wird erst der in **Stufe 1** (vgl. Abb. 3.1) beschriebene Algorithmus durchgeführt. Durch Zuordnung von Nachfragestandorten *i* zu den Angebotsstandorten *j* wird erreicht, daß alle Einzugsbereiche der Angebotsstandorte *j* die untere Kapazitätsbeschränkung (R_7) erfüllen.

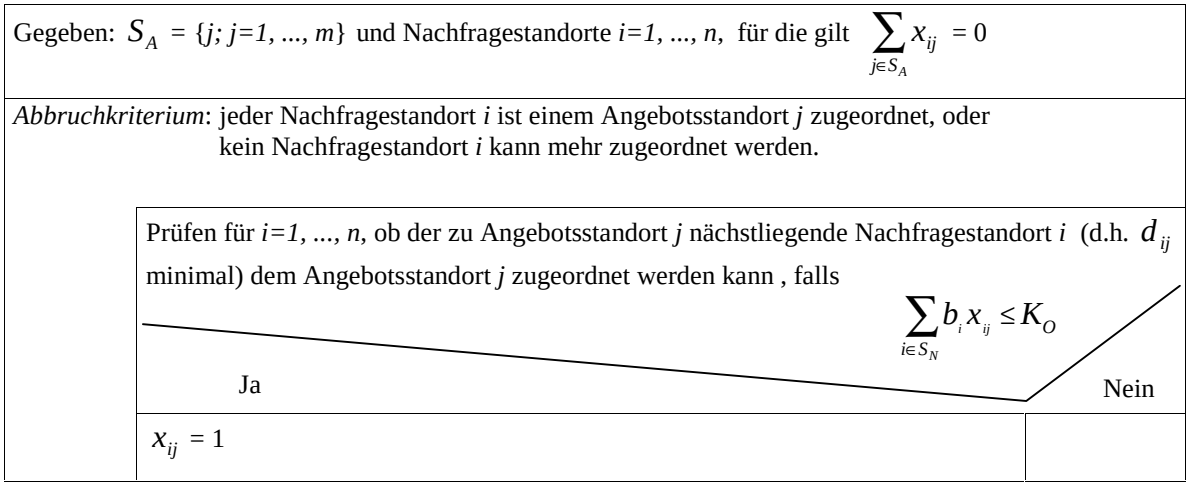
Abb. 3.1: Stufe 1 zur Erfüllung der unteren Kapazitätsbeschränkung



Nachdem dies erfolgreich geschehen ist, folgt der unter **Stufe 2** (vgl. Abb. 3.2) beschriebene Algorithmus. Hier wird die Menge der in Stufe 1 nicht zugeordneten Nachfragestandorte unter Einhaltung der geforderten Randbedingungen den Angebotsstandorten zugeordnet. Konnte Stufe 2 erfolgreich durchgeführt werden, so wurden alle Nachfragestandorte einem Angebotsstandort zugeordnet.

Für **Problemstellungen ohne untere Kapazitätsbeschränkung** wird Stufe 1 nicht durchgeführt, sondern sofort mit **Stufe 2** begonnen. In diesem Fall sind noch keine Nachfragestandorte *i* zu Angebotsstandorten *j* zugeordnet.

Abb. 3.2: Stufe 2 zur Zuordnung aller nicht zugeordneter Nachfragestandorte



Wenn es möglich war, alle Nachfragestandorte zuzuordnen, so ist das erste Teilverfahren abgeschlossen. Es gibt jedoch **zwei Fälle,** in denen das beschriebene Verfahren **keine zulässige Lösung** liefert, auch wenn eine solche existiert:

- Stufe 1 zur Erfüllung der unteren Kapazitätsbeschränkung (vgl. Abb.3.1) führt nicht zu Einzugsbereichen mit der notwendigen Mindestgröße. Ist die Startkonfiguration zufällig vorgegeben, kann mit einer anderen Zufallsmenge von Angebotsstandorten ein neuer Versuch unternommen werden. Waren alle Angebotsstandorte exogen vorgegeben (R_9), so gibt es für dieses System keine Lösung.

- Stufe 2 zur Zuordnung aller nicht zugeordneter Nachfragestandorte konnte nicht alle Nachfragestandorte unter Einhaltung der Obergrenze K_O zuordnen. Dann kommt ein **Austauschalgorithmus** zur Anwendung, der im wesentlichen versucht, bereits zugeordnete Nachfragestandorte mit noch nicht zugeordneten paarweise auszutauschen, bis eine vollständige Zuordnung möglich ist. Führt dieser Algorithmus nicht zum Ziel, wird wieder eine neue Startkonfiguration gewählt, falls diese nicht exogen vorgegeben waren.

Teilverfahren zur Optimierung des gefundenen Standort-Zuordnungssystems

Die im ersten Teilverfahren gewonnenen zulässigen Einzugsbereiche sollen so verändert werden, daß die für das spezifizierte Standortallokationsproblem gewählte Zielfunktion ein Minimum erreicht. Dies geschieht mittels eines Algorithmus, der drei Schritte beinhaltet, die so lange nacheinander durchgeführt werden, bis keine Verbesserung der Zielfunktion mehr möglich ist. Zur Illustration der einzelnen Schritte wird ein beispielhaftes Standort-Zuordnungssystem vor und nach der Durchführung des Algorithmusschrittes dargestellt.

1. Schritt: Für jeden Einzugsbereich von Angebotsstandort *j* wird geprüft, ob der Zielfunktionswert verbessert werden kann, wenn man den Angebotsstandort *j* durch einen zugeordneten Nachfragestandort *i* ersetzt. (vgl. Abb. 3.3, S. 27), immer unter Berücksichtigung der geforderten Randbedingungen. Wurde ein Austausch realisiert, so wird mit Schritt 2 weitergearbeitet, andernfalls mit Schritt 3.

2. Schritt: Überprüfen, ob durch Reallokation eines Nachfragestandorts *i* (*i=1, ..., n*) zu einem anderen Angebotsstandort *j* (*j=1, ..., m*) der Zielfunktionswert verbessert werden kann [unter Berücksichtigung der geforderten Randbedingungen] (vgl. Abb. 3.4, S. 27). Kann wenigstens ein Nachfragestandort *i* einem anderen Angebotsstandort *j* zugeordnet werden, wird wiederum mit dem 1. Schritt eine entsprechende Prüfung innerhalb des Einzugsbereiches durchgeführt, sonst setzt der Algorithmus mit Schritt 3 fort.

Abb. 3.3: Austausch zwischen Angebotsstandort *j* und einem zugeordneten Nachfragestandort *i*

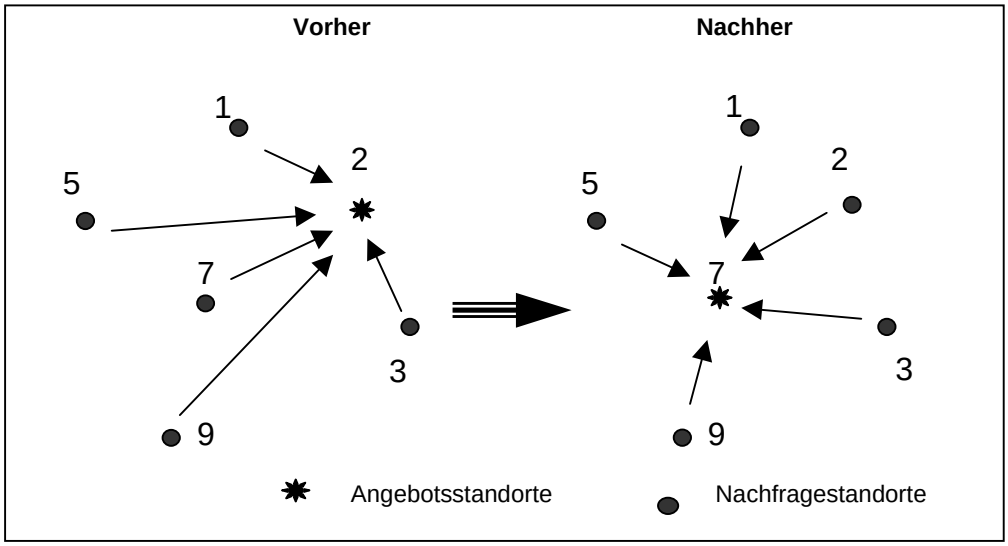
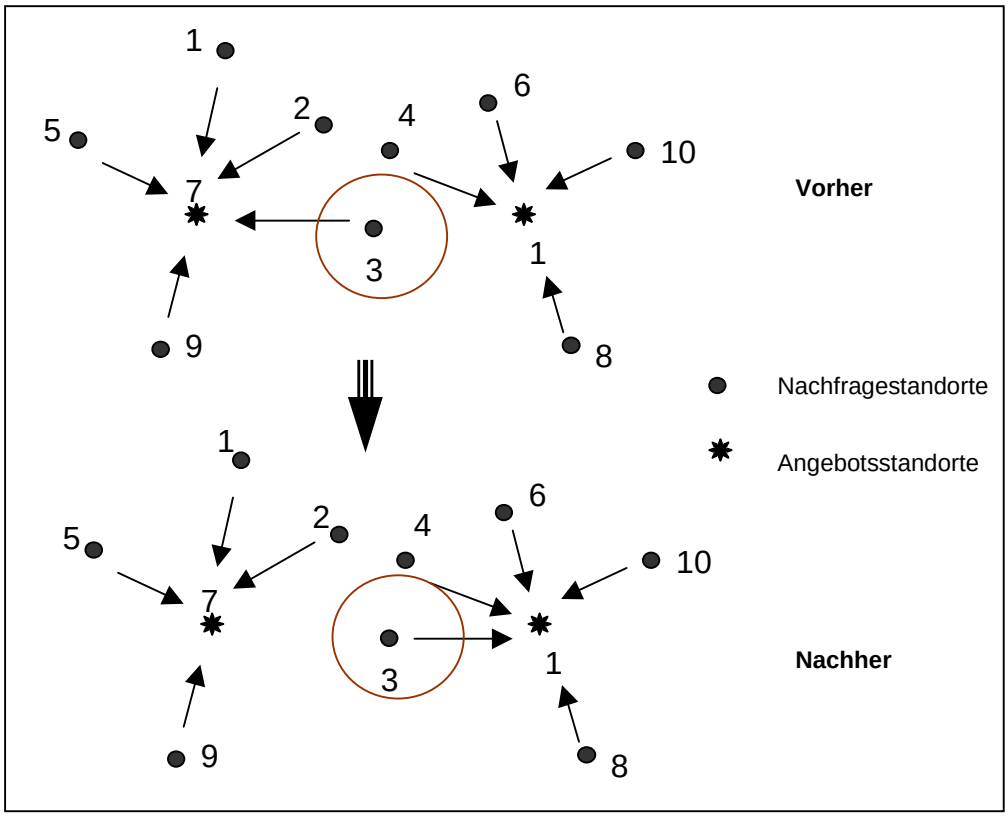
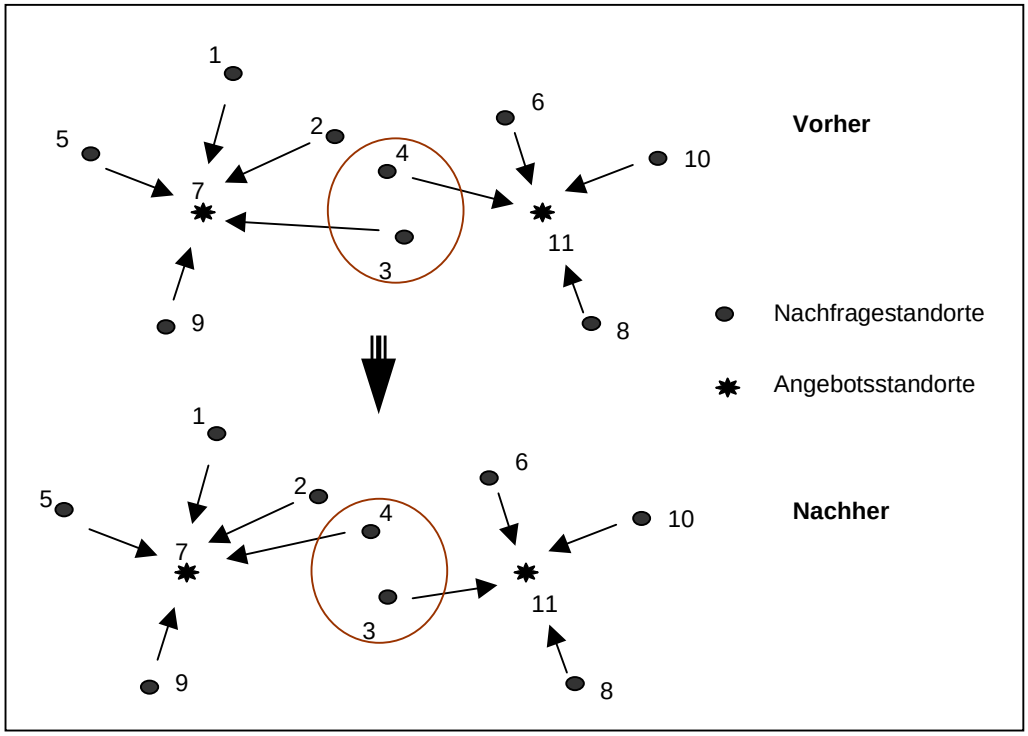


Abb. 3.4: Zuordnung eines Nachfragestandortes zu einem anderen Angebotsstandort



3. Schritt: Sei E_j der Einzugsbereich des Angebotsstandortes j mit $E_j = \{i,\ x_{ij} = 1\}$. Für alle Paare von Nachfragestandorten $(i_k,\ i_l)$, die in verschiedenen Einzugsbereichen liegen $(i_k \in E_k$ und $i_l \in E_l)$, wird überprüft, ob die Zuordnung von i_k zu E_l und von i_l zu E_k eine Verbesserung des Zielfunktionswertes mit sich bringen würde. Ist dies der Fall, wird der Austausch vorgenommen, immer unter Berücksichtigung der geforderten Randbedingungen (vgl. Abb. 3.5). Dadurch ergeben sich neue Einzugsbereiche, mit denen dann der Algorithmus wieder mit Schritt 1 startet.

Abb. 3.5: Paarweiser Austausch von Nachfragestandorten, die in verschiedenen Einzugsbereichen liegen

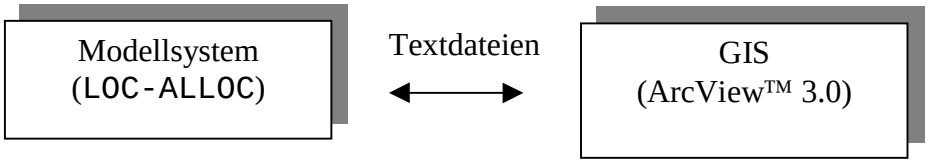


Das gesamte Verfahren wird so lange wiederholt, bis keine Verbesserungen mehr möglich sind. Dann wird geprüft, ob auch die Randbedingungen R_4 und R_5 ebenfalls erfüllt werden konnten. Ist dies nicht der Fall, so hat das heuristische Verfahren für diese Problemstellung mit der gewählten Startkonfiguration keine Lösung gefunden. Eine andere Startkonfiguration kann sehr wohl zu einer Lösung führen. Auch ist es möglich, daß eine andere Startkonfiguration bessere Optimierungsergebnisse liefern kann (Sensibilität des Verfahrens hinsichtlich der Startkonfigurationen).

3.3 Softwaretechnischer Ansatz

Der gewählte Ansatz basiert auf einer losen Koppelung des Modellsystems (LOC-ALLOC) mit dem GIS ArcView™ 3.0 (vgl. FISCHER 1996). Der Datenaustausch zwischen den beiden Systemen erfolgt ausschließlich über Textdateien (vgl Abb.3.6). Zusätzlich wurde eine graphische Benutzeroberfläche (GUI-LOC-ALLOC) entwickelt, um dem Benutzer, die im Anforderungsprofil geforderten Funktionalitäten zur Verfügung stellen zu können.

Abb. 3.6: Lose Koppelung des Modellsystems mit einem GIS (vgl. FISCHER 1996, S. 12)



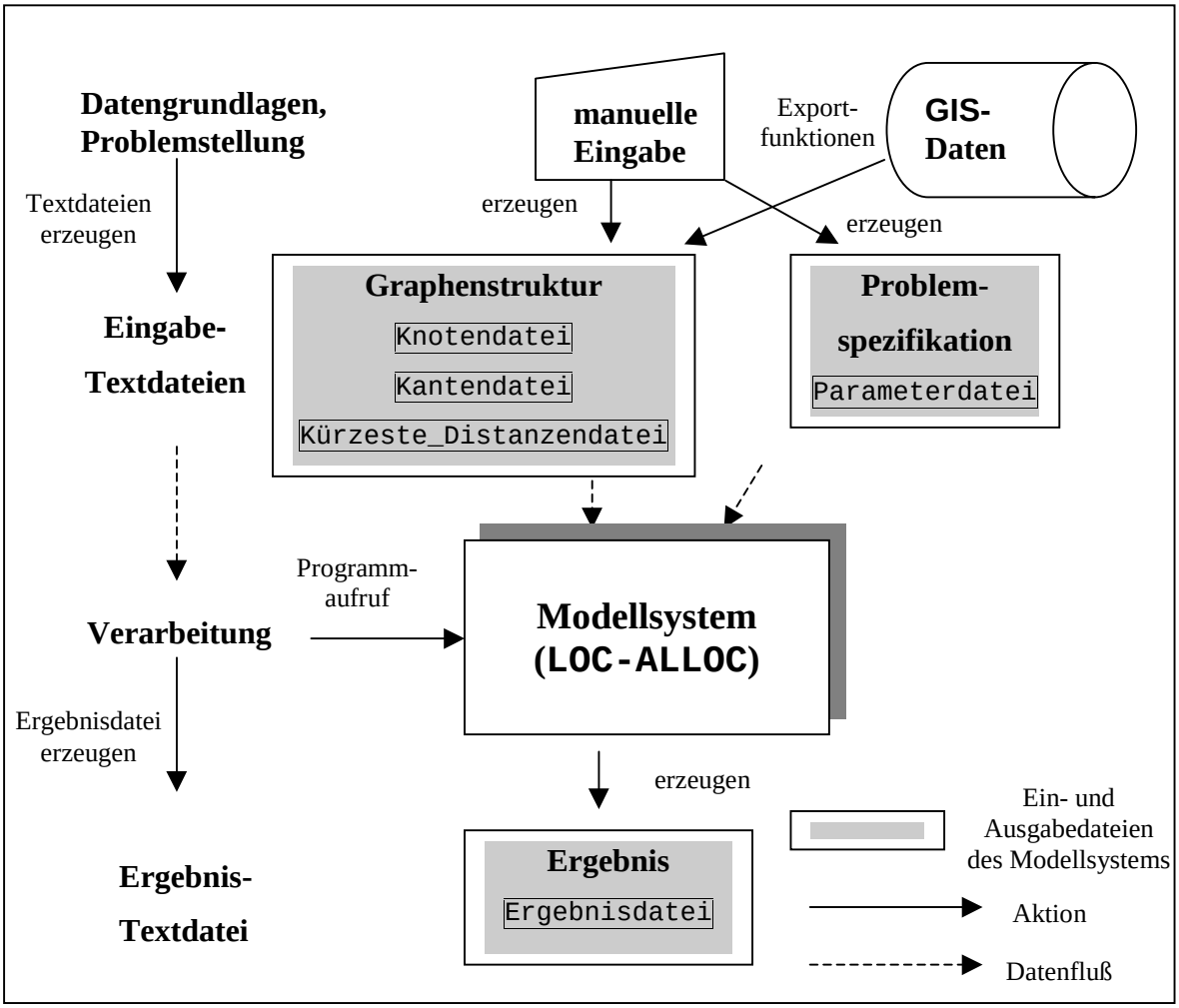
Modellsystem (LOC-ALLOC)

Der in 3.2 beschriebene heuristische Ansatz wird in einer objektorientierten Programmiersprache (C++) implementiert. Dadurch ergibt sich eine hohe Flexibilität im Hinblick auf eine funktionale Erweiterung des Modellsystems. Als Schnittstelle zur Dateneingabe und –ausgabe werden **Textdateien** verwendet. Das in Kapitel 3.2 beschriebene heuristische Verfahren benötigt einerseits die Struktur des Graphen, die in drei Dateien abgespeichert ist (Knotendatei, Kantendatei, Kürzeste_Distanzendatei). Andererseits muß die Spezifikation des Standortallokationsproblems für die Modellrechnung zur Verfügung gestellt werden. Dies erfolgt über die Erstellung einer Parameterdatei. Sie enthält eine a priori Festlegung der Parameterwerte des jeweiligen Zuordnungsproblems. Mit diesen vier Eingabedateien berechnet das Modellsystem die Lösung des in der Parameterdatei spezifizierten Standortallokationsproblems. Das Modellergebnis wird in der Ergebnisdatei abgespeichert.

Im Flußdiagramm in Abb. 3.7 ist der Ablauf der Modellrechnung und das Zusammenspiel zwischen Modellsystem und Textdateien dargestellt. Ausgangsbasis sind die **Datengrundlagen** und die **Problemstellung**.

Bei den Datengrundlagen kann es sich entweder um GIS-Daten handeln oder um einen Graphen, der in einer bestimmten Form vorliegt (dies kann beispielsweise eine Zeichnung sein, können aber auch Matrizen sein). Diese Daten müssen in **Textdateien** umgewandelt werden, die vom Modellsystem verarbeitet werden können. Für GIS-Daten kann dies über Exportfunktionen geschehen. Liegen die Daten in anderer Form vor, so müssen sie entweder programmtechnisch transformiert werden oder sie sind manuell einzugeben. Die Problemstellung hat durch die Spezifikation der jeweiligen Parameterwerte für Randbedingungen und Zielfunktionen in der `Parameterdatei` zu erfolgen. Das Modellsystem wird aufgerufen, liest die Daten aus den Eingabedateien, führt die Berechnungen durch und erzeugt eine `Ergebnisdatei`.

Abb. 3.7: Ablauf einer Modellrechnung und das Zusammenspiel zwischen Modellsystem (LOC-ALLOC) und Textdateien



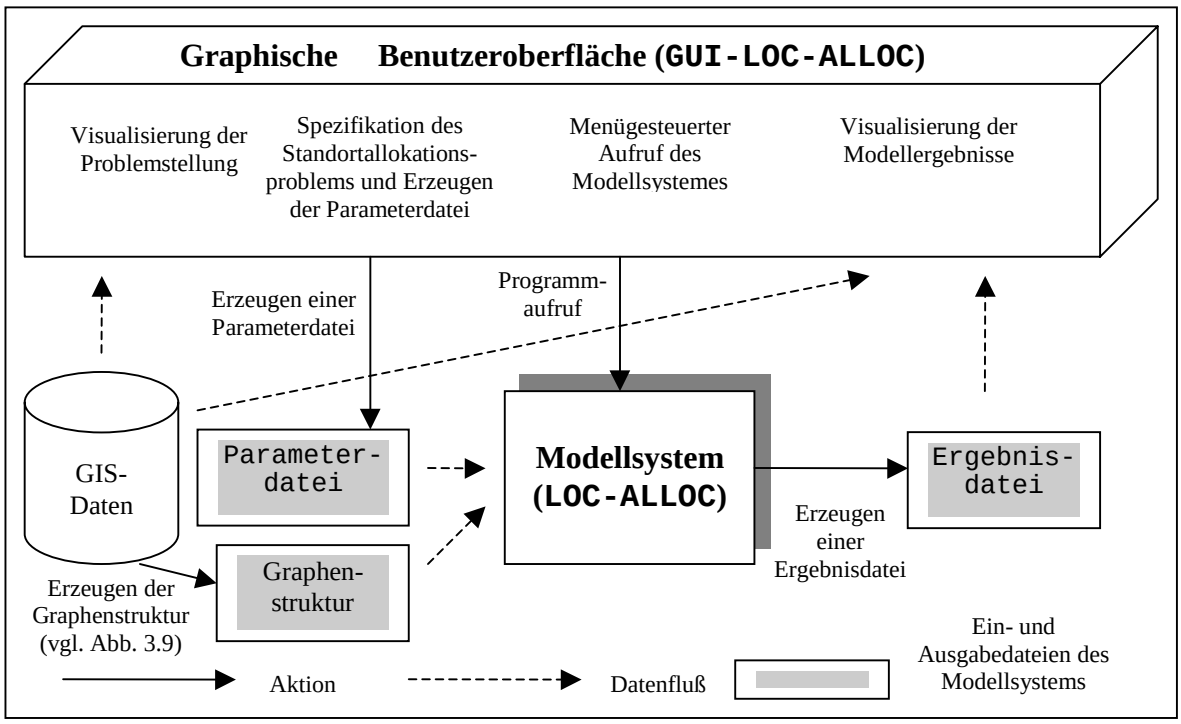
Graphische Benutzeroberfläche (GUI-LOC-ALLOC)

Die graphische Benutzeroberfläche soll drei grundlegende Funktionalitäten für den Benutzer zur Verfügung stellen:

1) Visualisierung der Problemstellung und der Modellergebnisse, sowie Darstellung wichtiger statistischer Kennzahlen für das gefundene Standort-Zuordnungssystem,
2) Spezifikation des Standortallokationsproblems über Menüleiste oder durch graphische Auswahl und Erzeugen einer Parameterdatei, die die Problemspezifikation enthält; zur Unterstützung stehen GIS-Basisfunktionen zur Verfügung,
3) Menügesteuerter Aufruf des Modellsystems mit Angaben über zu verwendende Parameterdatei und Graphendateien.

Für 1) ist es notwendig, die Problemstellung auch im GIS-Datenformat von **ArcView**™ vorliegen zu haben. Abb. 3.8 zeigt das Zusammenspiel von graphischer Benutzeroberfläche (GUI-LOC-ALLOC), Modellsystem (LOC-ALLOC) und Textdateien.

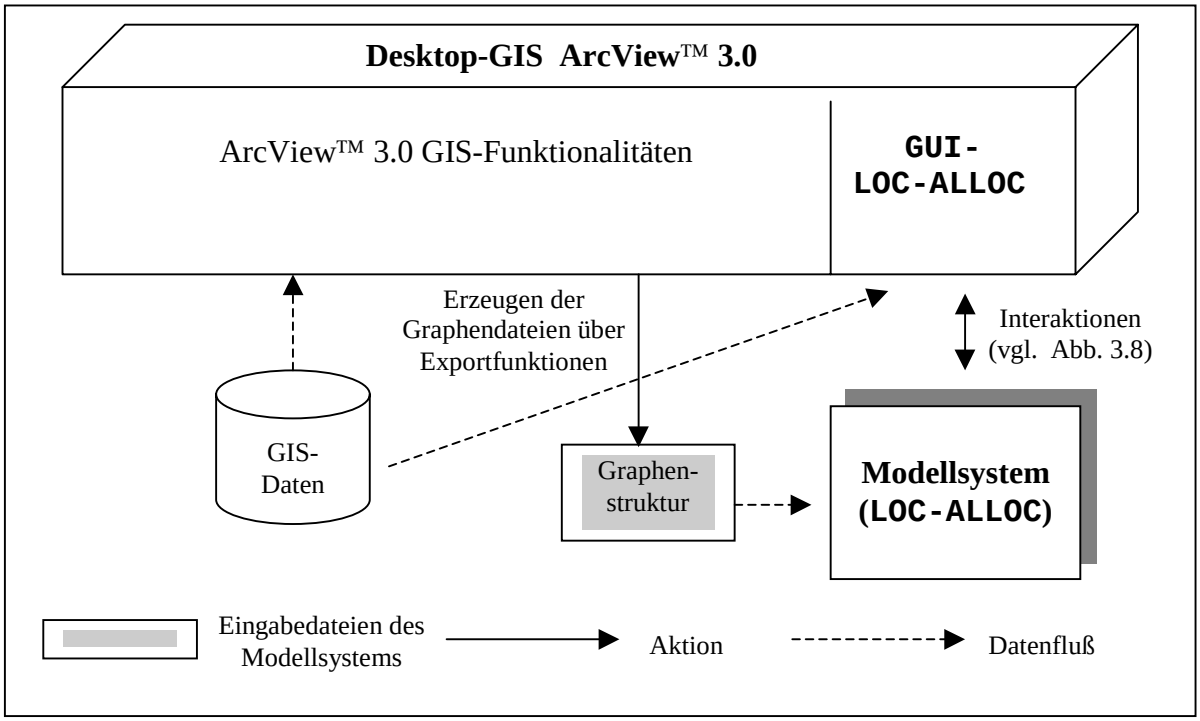
Abb. 3.8: Zusammenspiel von graphischer Benutzeroberfläche (GUI-LOC-ALLOC), Modellsystem (LOC-ALLOC) und Textdateien



Integration der graphischen Benutzeroberfläche (GUI-LOC-ALLOC) in des Desktop-GIS ArcView™

Da GUI-LOC-ALLOC mit AVENUE™, der Programmiersprache des Desktop-GIS ArcView™ 3.0, entwickelt wurde, ist es möglich, GUI-LOC-ALLOC in das Desktop-GIS ArcView™ 3.0 zu integrieren. Damit stehen die Analysefunktionen von LOC-ALLOC innerhalb des GIS in einer einheitlichen Benutzeroberfläche zur Verfügung. Zur Lösung von Standortallokationsproblemen ist es aber nach wie vor notwendig, die von LOC-ALLOC benötigten drei Dateien zur Graphenspezifikation zu erzeugen. Es können also nicht direkt GIS-Daten in das Modellsystem als Eingangsdaten übernommen werden, aber es besteht die Möglichkeit, über die Exportfunktionen von ArcView™ in Verbindung mit dem Network Analyst™ die erforderlichen Dateien zu erzeugen. Die Informationen über den exakten Aufbau dieser Dateien sind im Anhang A zu finden. Es handelt sich also nicht um eine vollständige Integration des Modellsystems in ein GIS, sondern lediglich um die Integration der Benutzeroberfläche (siehe Abb. 3.9).

Abb. 3.9: Integration der graphischen Benutzeroberfläche (GUI-LOC-ALLOC) in das Desktop-GIS ArcView™



4. Einführung in die Arbeit mit LOC-ALLOC

LOC-ALLOC ist ein Programmpaket zur Lösung von Standortallokationsproblemen mit Hilfe von Modellen des elementaren Modelltyps 2, 3 und 5 oder einer Kombination dieser Modelltypen (vgl. Kapitel 2.3). Die graphische Benutzerschnittstelle (GUI-LOC-ALLOC) ist in der Makrosprache AVENUE™ des Desktop-GIS ArcView™ 3.0 programmiert. In diesem Kapitel soll der praktische Einsatz von LOC-ALLOC mit der graphischen Benutzerschnittstelle (GUI-LOC-ALLOC) demonstriert werden. In 4.1 wird anhand von zwei **Übungsbeispielen** die Funktionsweise des Softwarepaketes (LOC-ALLOC mit GUI-LOC-ALLOC) beschrieben. In 4.2 soll ein realweltliches Problem betrachtet und die Lösung beschrieben werden. Das realweltliche Beispiel bezieht sich auf eine Neueinteilung der Postamtsbezirke des südwestlichen Niederösterreichs, etwa um entsprechende Kosten im Kontext der Rationalisierung der Post einsparen zu können.

4.1 Übungsbeispiele mit der graphischen Benutzeroberfläche (GUI-LOC-ALLOC)

Als Übungsbeispiel dient ein Graph mit 30 Knoten (vgl. BAHRENBERG 1979), an dem die Funktionsweise des Softwarepaketes veranschaulicht wird. Vor der Durchführung der Übungen ist sicherzustellen, daß LOC-ALLOC und die Beispieldaten ordnungsgemäß installiert wurden (vgl. Anhang A.2).

Übung 1: Arbeit mit LOC-ALLOC

Inhalt dieser Übung:

1) Aufruf des Programmpaketes,
2) Wahl einer Benutzeroberfläche,
3) Laden eines Übungsbeispieles und Visualisierung der Problemstellung,
4) Modellrechnung zur Lösung des gegebenen Standortallokationsproblems,
5) Visualisierung der Angebotsstandorte (supply locations) und der Zuordnungen der Nachfragestandorte (service areas) zu den Angebotsstandorten,
6) Darstellung der Zielfunktionswerte und der Nachfragewerte für die sechs Angebotsstandorte in Form von Balkendiagrammen,
7) Löschen der Ergebnisse aus dem Arbeitsspeicher zur Durchführung weiterer Modellrechnungen.

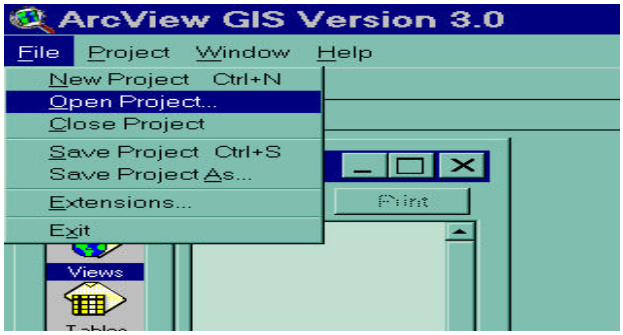
1) Aufruf des Programmpaketes

Da es sich bei der graphischen Benutzeroberfläche um ein programmiertes *Project* des Desktop-GIS ArcView™ 3.0 handelt, muß erst dieses Programm gestartet werden.

💻 Starten von ArcView™ 3.0

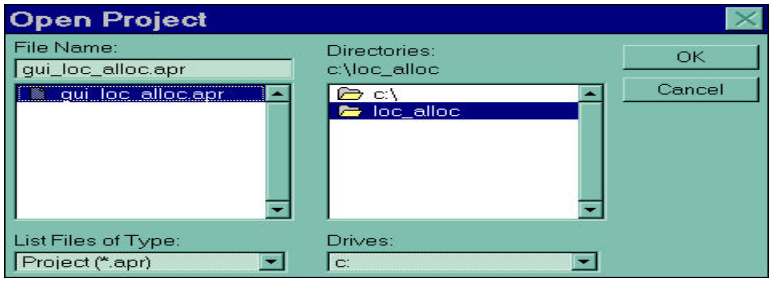
💻 Unter dem Menüpunkt `File` wählen Sie `Open Project`.

Abb. 4.1: Öffnen des Dialogfensters zum Laden der graphischen Benutzeroberfläche (`GUI-LOC-ALLOC`) unter ArcView™



💻 Laden Sie das Projekt `gui_loc_alloc.apr` aus dem Verzeichnis `loc_alloc`

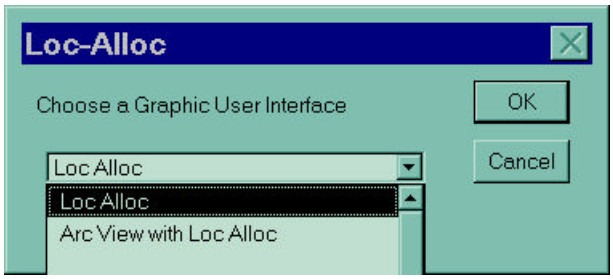
Abb. 4.2: Laden der graphischen Benutzeroberfläche (`GUI-LOC-ALLOC`)



2) Wahl einer Benutzeroberfläche

- Wenn Sie mit ArcView™ nicht vertraut sind, so wählen Sie, wie in diesem Beispiel `Loc Alloc`.

Abb. 4.3: Wahl einer Benutzeroberfläche



3) Laden eines Übungsbeispieles und Visualisierung der Problemstellung

- Unter dem Menüpunkt `Location Criteria` wählen Sie `Load Specification`, um das Übungsbeispiel `demo_specification` zu laden.

Abb. 4.4a: `Location Criteria` Pull Down Menü

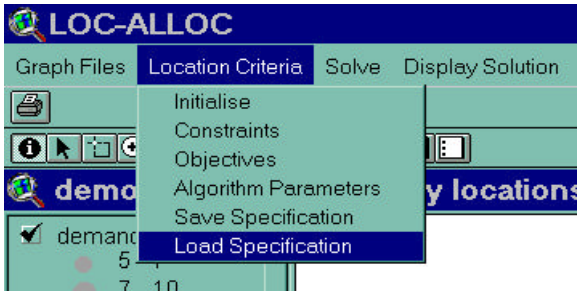
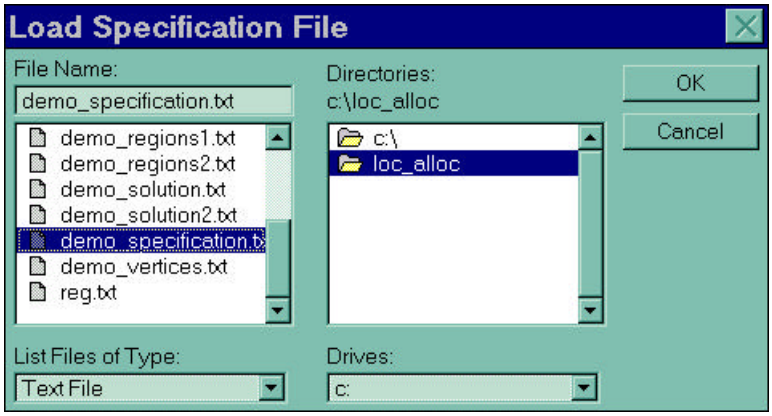
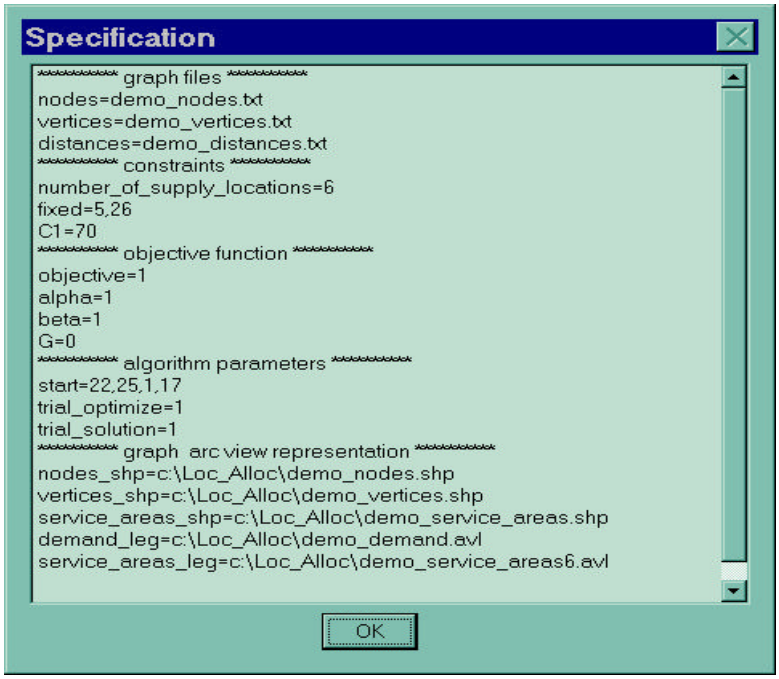


Abb. 4.4b: Laden des Übungsbeispieles `demo_specification`



Der Inhalt der Datei `demo_specification` wird angezeigt (vgl. Abb. 4.5). Sie enthält die Spezifikation des in dieser Übung gegebenen Standortallokationsproblems und entspricht der in Kapitel 3.3 eingeführten `Parameterdatei`. Bei der Beschreibung des Standortallokationsproblems wird kurz in eckigen Klammern [] auf die entsprechenden Parameterwerte hingewiesen. Die genauen Definitionen und die Bedeutung der einzelnen Parameter wird in Anhang A.3 erklärt.

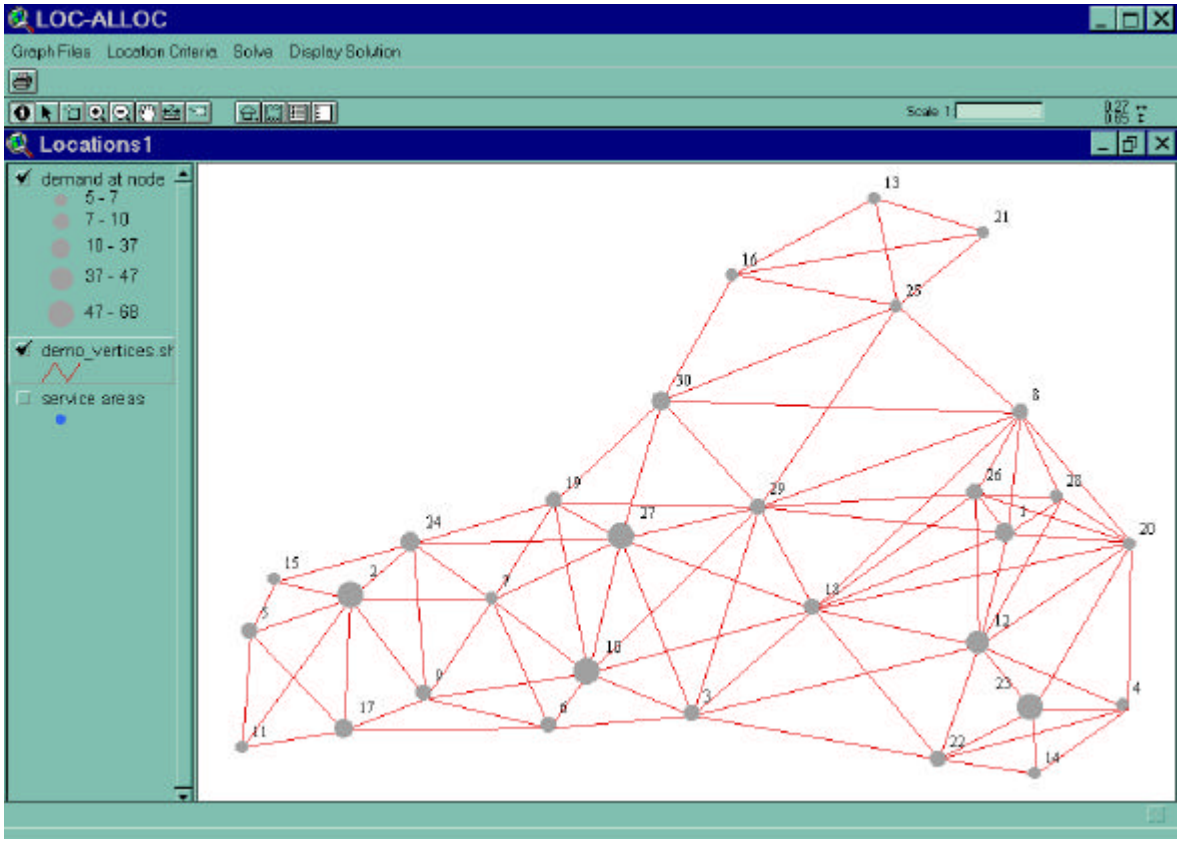
Abb. 4.5: Inhalt der Datei `demo_specification`



Problemstellung des Übungsbeispieles: Gegeben sei eine Menge von 30 Nachfragestandorten [*nodes=demo_nodes*] mit Kantenverbindungen [*vertices=demo_vertices*], wie in Abb. 4.6 dargestellt. Die Matrix (d_{ij}) der kürzesten Wege zwischen diesen Standorten ist gegeben [*distances=demo_distances*]. Die Nachfragestandorte 5 und 26 sind exogen als Angebotsstandorte vorgegeben [*fixed=5,26*]. Die restlichen Standorte sind Nachfragestandorte und potentielle Angebotsstandorte. Es sollen 6 Angebotsstandorte [*number_of_supply_locations=6*] mit Einzugsbereichen ermittelt werden, die die geforderten Randbedingungen [*Constraints*] erfüllen. Die Zielfunktion Z_1 (maximale Distanz) soll minimiert werden [*objective=1*].

- Bei der folgenden Abfrage (`Check Actuality`) wird bestätigt, daß diese Parameter verwendet werden sollen. Nach der Bestätigung werden die Parameter geladen und die graphische Repräsentation des Graphen angezeigt (vgl Abb. 4.6). Jeder Nachfragestandort hat eine Nummer und in der Legende wird die Nachfrage an den Standorten durch verschiedene Punktgrößen angegeben (*demand at node*). Über das Symbol  erhalten sie einen Ausdruck des Fensters.

Abb. 4.6: Graph zu Übungsbeispiel 1



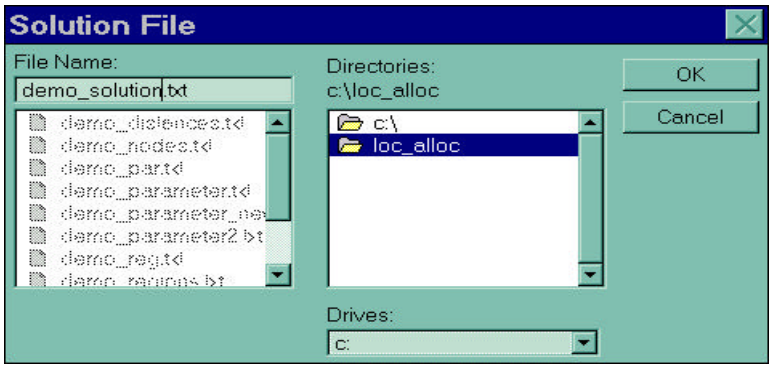
4) Modellrechnung zur Lösung des gegebenen Standortallokationsproblems

- Unter dem Menüpunkt `Solve` wählen Sie `Start Computation`. (vgl. Abb. 4.7).

Abb. 4.7: Starten der Modellrechnung

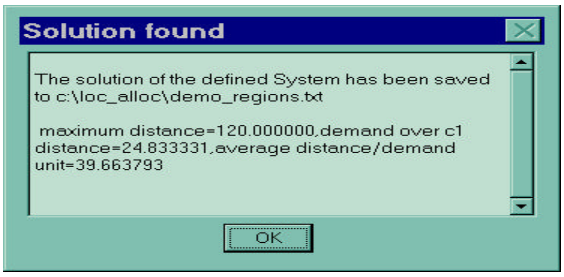
- Sie werden aufgefordert einen Dateinamen für die Ergebnisdatei (*Solution File*) einzugeben (vgl. Abb. 4.8). Geben Sie `demo_solution.txt` ein und bestätigen Sie mit `OK`.

Abb. 4.8: Dialogfenster zur Eingabe eines Dateinamens für die `Ergebnisdatei`



Während die Berechnungen durchgeführt werden, zeigt ein Kommunikationsfenster den Status `working` an. Anschließend wird eine Nachricht angezeigt, daß die Berechnung beendet wurde. Nach der Bestätigung mit `OK` wird das Ergebnis für die 3 Zielfunktionen angezeigt (vgl. Abb. 4.9). Definitionen und Bedeutung der Zielfunktionenswerte wird in Punkt 6 dieser Übung in Zusammenhang mit ihren Diagrammen erklärt. Sollte ein Fehler aufgetreten sein, dann gibt es eine Fehlermeldung, die auch die Art des Fehlers beinhaltet.

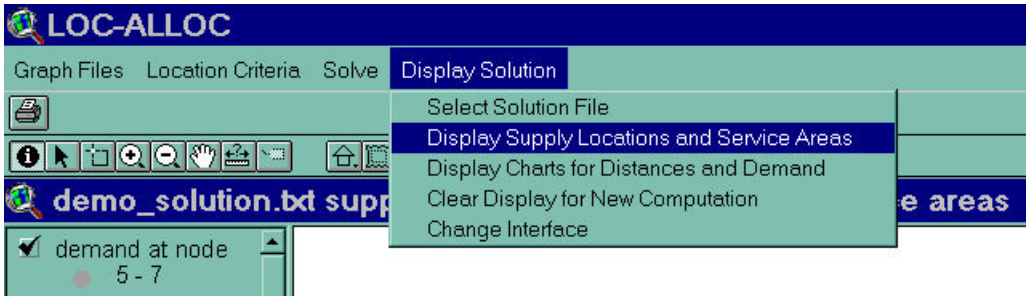
Abb. 4.9: Dialogfenster Ergebnisausgabe



5) Visualisierung der Angebotsstandorte (supply locations) und der Zuordnungen der Nachfragestandorte zu den Angebotsstandorten (service areas)

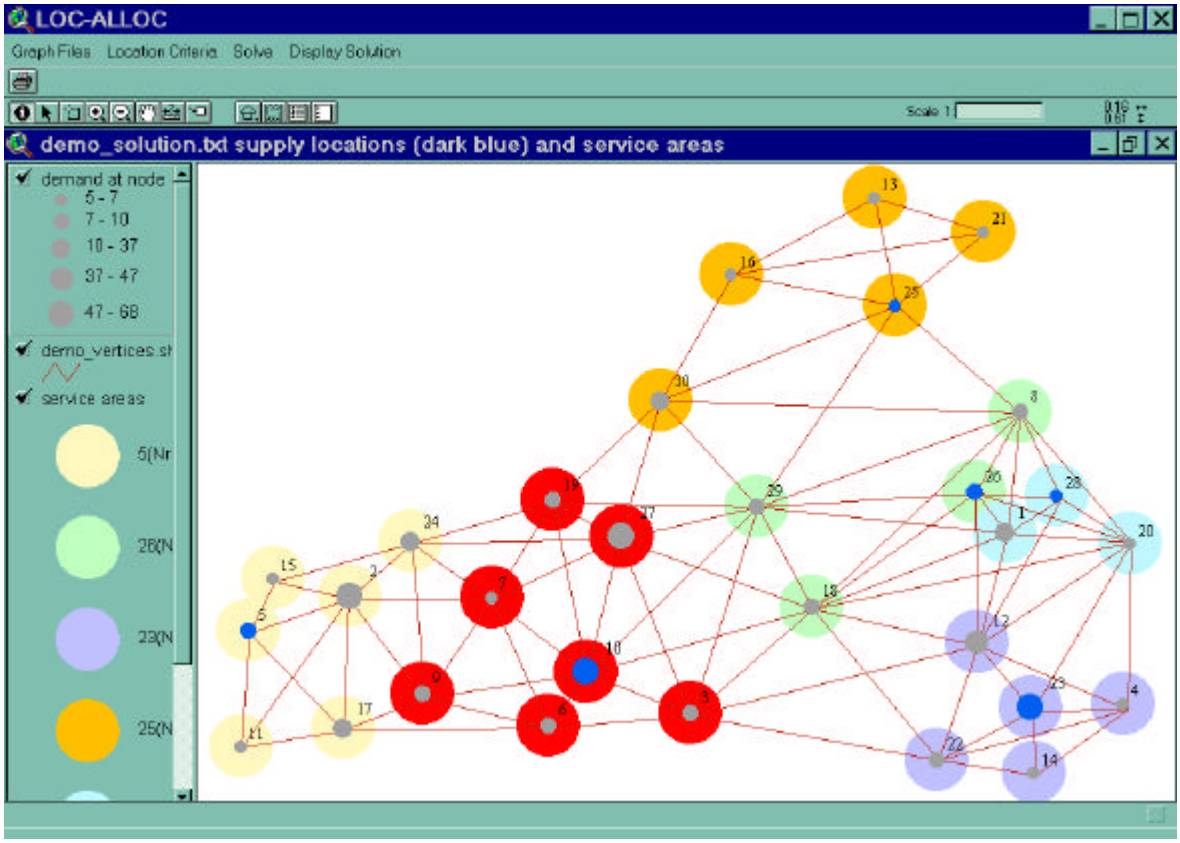
- Unter dem Menüpunkt `Display Solution` wählen Sie `Display Supply Locations and Service Areas` (vgl. Abb. 4.10).

Abb. 4.10: `Display Solution` Pull Down Menü



Am Bildschirm werden die Angebotsstandorte (supply locations) dunkelblau markiert. Die Zuordnungen der Nachfragestandorte zu den Angebotsstandorten (service areas) werden in einheitlicher Farbe dargestellt (vgl. Abb. 4.11).

Abb. 4.11: Angebotsstandorte (supply locations) und Zuordnungen der Nachfragestandorte zu den Angebotsstandorten (service areas): Übungsbeispiel 1

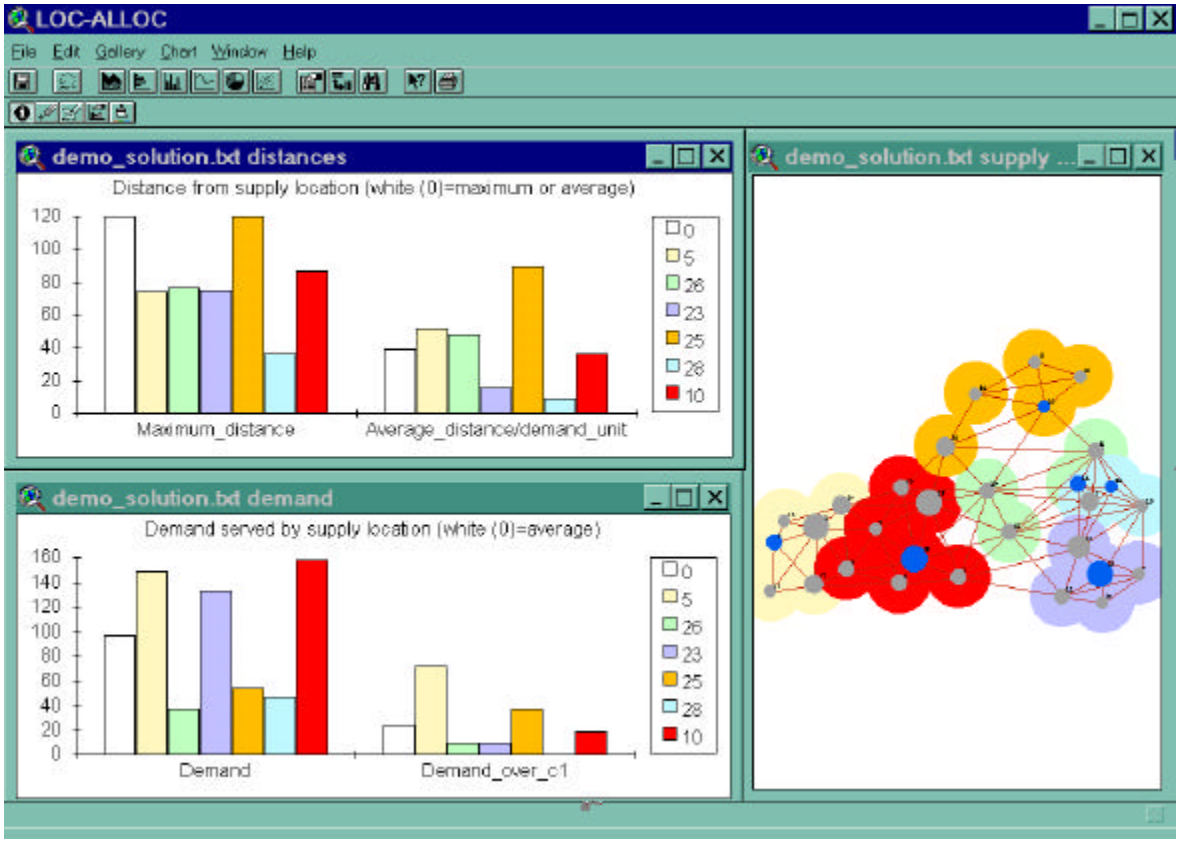


Die Legende beinhaltet unter *service areas* die Nummer des Angebotsstandortes (vgl. Abb. 4.6) für die in dieser Farbe ausgewiesenen Nachfragestandorte.

6. Darstellung der Zielfunktionswerte und der Nachfragewerte für die sechs Angebotsstandorte in Form von Balkendiagrammen

- Zur Charakterisierung der Modellergebnisse (diskrete Zuordnungen von Nachfragestandorten zu den Angebotsstandorten) anhand der Zielfunktionswerte und der Nachfragewerte für die einzelnen Angebotsstandorte, müssen Sie unter dem Menüpunkt `Display Solution` die Option `Display Charts for Distances and Demand` wählen. Am Bildschirm werden zwei Fenster mit vier Diagrammen angezeigt (vgl. Abb. 4.12).

Abb. 4.12: Darstellung der Zielfunktionswerte und der Nachfragewerte für die sechs Angebotsstandorte in Form von Balkendiagrammen: Übungsbeispiel 1



Jedes Diagramm enthält auf der x-Achse für die sechs Angebotsstandorte je einen Balken und einen weißen, der den Durchschnittswert der sechs Angebotsstandorte angibt (Ausnahme: im Diagramm **Maximum_Distance** wird der Maximalwert angegeben). In der Legende ist der weiße Balken mit Null (0) bezeichnet und die sechs Angebotsstandorte mit der ihnen zugewiesenen Nummer (vgl. Abb. 4.6). Die Farben der Balken in den

Diagrammen stimmen mit den Farben der jeweiligen Einzugsbereiche in Abb. 4.11. überein.

Das Fenster für die **Nachfrage** (**`demo_solution.txt demand`**) enthält zwei Diagramme. In beiden Diagrammen wird auf der y-Achse die Nachfrage in Nachfrageeinheiten (*demand_unit*, z.B.: Einwohner) angezeigt. Das Diagramm **`Demand`** enthält für jeden Angebotsstandort, die von ihm in den zugeordneten Nachfragestandorten versorgte Nachfrage. Im Diagramm **`Demand_over_C1`** wird nur jene Nachfrage ausgewiesen, die oberhalb der Transportaufwandsobergrenze C_1 liegt. Dieser Wert entspricht dem Zielfunktionswert Z_2 (vgl. Kapitel 2, S. 16).

Das Fenster für die **Transportkosten** (**`demo_solution.txt distances`**) enthält ebenfalls zwei Diagramme. In beiden Diagrammen werden auf der y-Achse die Transportkosten in Transportkosteneinheiten (z.B.: Kilometer oder Schilling) angezeigt. Das Diagramm **`Maximum_Distance`** enthält für jeden Angebotsstandort die Transportkosten für den von ihm am weitesten entfernten zugeordneten Nachfragestandort. Dieser Wert entspricht dem Zielfunktionswert Z_1 (vgl. Kapitel 2, S. 16). Das Diagramm **`Average_distance/demand_unit`** enthält einen Vergleichswert für das Produkt (möglicherweise gewichtet) von Nachfrage und Transportkosten, dessen Minimierung in Zielfunktion Z_3 angestrebt wird (vgl. Kapitel 2, S. 17). Dieser Vergleichswert entspricht den durchschnittlichen Transportkosten (*Average_distance*), die pro Nachfrageeinheit (*demand_unit*, z.B.: Einwohner) aufgewendet werden müssen. Ein kleines Beispiel soll diesen Wert verdeutlichen:

> Existiert eine Nachfrage von 100 Einwohnern am Nachfragestandort *i* und die Transportkosten vom Angebotsstandort *j* zum Nachfragestandort *i* betragen 30.000 Schilling, so belaufen sich die durchschnittlichen Transportkosten pro Nachfrageeinheit (Einwohner) auf 300 Schilling. Existiert aber eine Nachfrage von 1000 Einwohnern am Nachfragestandort *i* bei gleichbleibenden Transportkosten, so betragen die durchschnittlichen Transportkosten pro Nachfrageeinheit (Einwohner) nur 30 Schilling.

Über das Symbol  erhalten sie einen Ausdruck des jeweiligen Fensters.

7) Löschen der Ergebnisse aus dem Arbeitsspeicher zur Durchführung weiterer Modellrechnungen.

💻 Um weitere Berechnungen durchführen zu können, müssen die Darstellung der Angebotsstandorte mit ihren Zuordnungen und die Balkendiagramme aus dem Arbeitsspeicher gelöscht werden. Dies geschieht mit dem Menüpunkt `Display Solution` und der Option `Clear Display for New Computation`. Der Ausgangszustand, der nach `Load Parameters` geherrscht hat, wird wieder hergestellt. Die Ergebnisse der Modellrechnung sind weiterhin unter dem Dateinamen der Ergebnisdatei (hier: `demo_solution.txt`)abgespeichert und können jederzeit mit dem Menüpunkt `Display Solution` und der Option `Select Solution File` wieder geladen werden.

Übung 2: Formulierung eines Standortallokationsproblems mit Kapazitätsbeschränkungen

Inhalt dieser Übung:

1) Spezifikation von Randbedingungen über ein Menü,
2) Graphische Spezifikation von Randbedingungen,
3) Visualisierung von Randbedingungen,
4) Spezifikation der Zielfunktion,
5) Erzeugung einer Parameterdatei für das spezifizierte Standortallokationsproblem

Für diese Übung können die Ausgangsdaten der Übung 1 verwendet werden (vergewissern Sie sich, daß die Ergebnisse aus dem Arbeitsspeicher gelöscht wurden (Übung 1: Punkt 7)). Müssen die Ausgangsdaten neu geladen werden, so geschieht dies wie in Übung 1 unter Punkt **3 Laden eines Übungsbeispieles und Visualisierung der Problemstellung** beschrieben.

1) Spezifikation von Randbedingungen über ein Menü

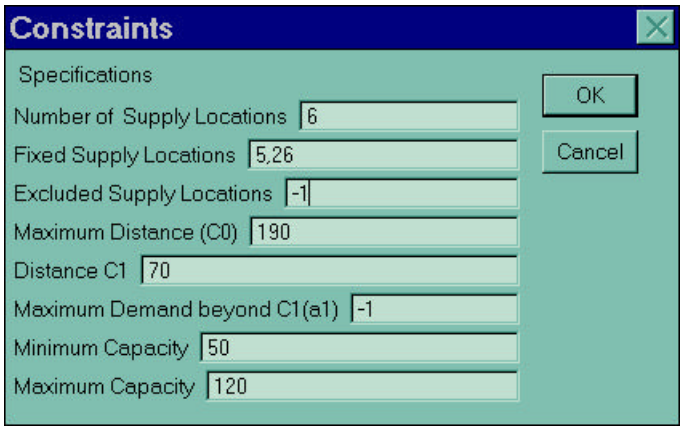
Zusätzlich zu den Randbedingungen in Übung 1 sollen in dieser Übung Kapazitätsbeschränkungen und eine Mindestanforderung an die Erreichbarkeit in die

Analyse einbezogen werden. Eine Kapazitätsuntergrenze (R_7) von 50 Nachfrageeinheiten (K_U=50), eine Kapazitätsobergrenze (R_8) von 120 Nachfrageeinheiten (K_O=120) und ein maximaler Transportaufwand (R_4) von 190 Transportkosteneinheiten (C_0=190). Dies geschieht unter dem Menüpunkt `Location Criteria` mit der Option `Constraints`. Geben Sie in den jeweiligen Zeilen die folgenden Werte ein:

- Für die Kapazitätsuntergrenze (R_7): *Minimum Capacity:* 50
- Für die Kapazitätsobergrenze (R_8): *Maximum Capacity:* 120
- Für die maximale Distanz (R_4): *Maximum Distance (C_0):* 190

und drücken sie `OK` (vgl. Abb. 4.13).

Abb. 4.13: Menü zur Spezifikation von Randbedingungen



2) Graphische Spezifikation von Randbedingungen

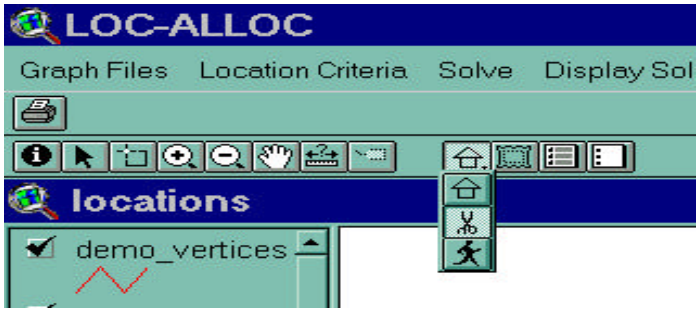
Weiters sollen acht Nachfragestandorte als Angebotsstandorte ausgeschlossen werden (R_{10}). In dieser Übung soll die Menge S_0 = {4, 7, 13, 14, 16, 20, 21, 28} der Nachfragestandorte als Angebotsstandorte ausgeschlossen werden. Dies könnte ebenfalls unter dem in Punkt 1 verwendeten Menüpunkt geschehen, soll jetzt aber direkt durch eine graphische Spezifikation erfolgen. Dazu müssen drei Schritte durchgeführt werden:

2a) Wahl der Randbedingung (Ausgeschlossene Angebotsstandorte (R_{10}))

Mit dem Symbol  (*specify set of excluded supply locations*) aus dem

Werkzeugmenü (vgl. Abb. 4.15) geben Sie an, daß Sie die Menge der ausgeschlossenen Angebotsstandorten spezifizieren wollen.

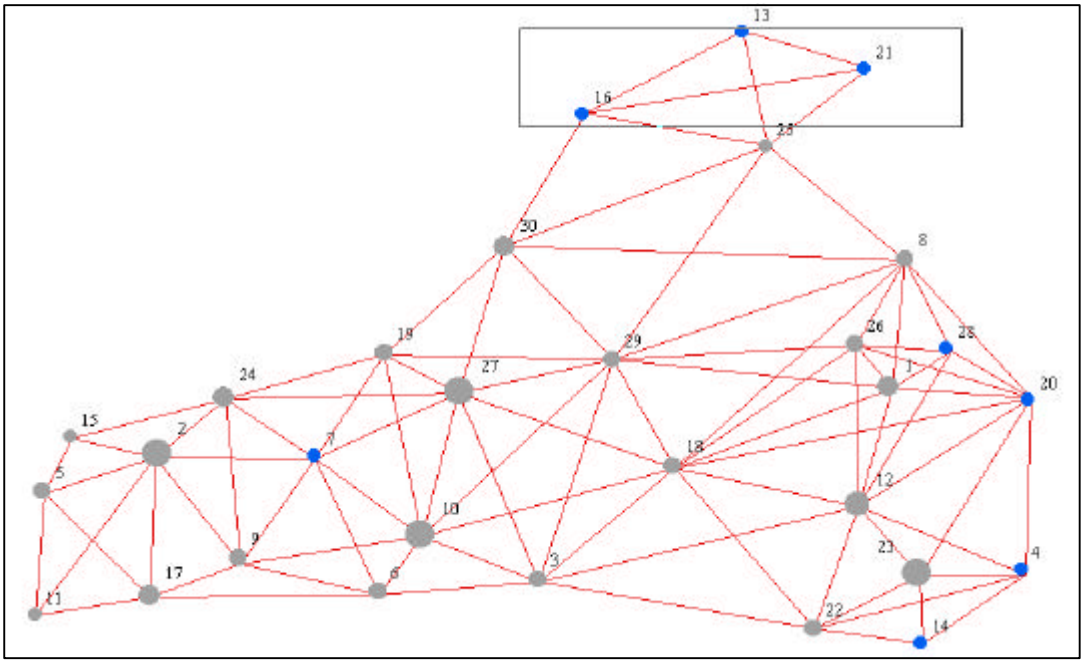
Abb. 4.14: LOC-ALLOC Werkzeugleiste mit Werkzeugmenü



2b) Markierung der betroffenen Nachfragestandorte:

- Durch Klicken auf das Selektions-Tool ( ) erscheint der Cursor als Rechteck. Mit Hilfe dieses Rechtecks können Standorte ausgewählt werden, die dann dunkelblau erscheinen (vgl. Abb. 4.14).

Abb. 4.15: Graphische Spezifikation der Randbedingung



Da mit einem Rechteck nicht alle auszuschließenden Nachfragestandorte gleichzeitig markiert werden können, muß die Auswahl nacheinander erfolgen. Hält

man bei der Benützung des Werkzeugs die Shift-Taste gedrückt, so werden die gewählten Standorte zu den schon markierten hinzugefügt. Sonst werden immer nur jene markiert, die innerhalb des Rechteckes liegen. Auf diese Art sollen die acht Standorte markiert werden.

2c) Spezifikation der markierten Nachfragestandorte als ausgeschlossene Angebotsstandorte (S_0)

Um die markierten Standorte in die Menge der ausgeschlossenen Angebotsstandorte (S_0) aufzunehmen, wird das Symbol  (*enter selected locations as set of specified*) angeklickt.

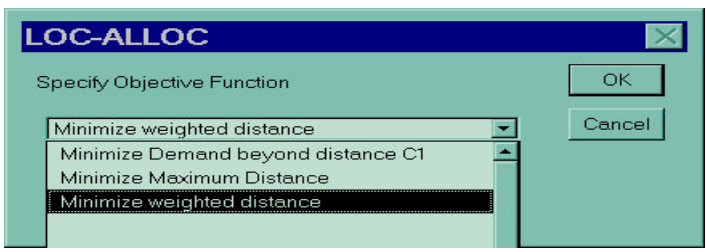
3) Visualisierung von Randbedingungen

- Will man überprüfen, ob beispielsweise die Menge $S_1 = \{5, 26\}$ der vorgegebenen Angebotsstandorte den spezifizierten entspricht (vgl. Übung 1), so muß bei der Wahl der Randbedingung (vgl. Punkt 2a) das Symbol  (*specify set of fixed supply locations*) angeklickt werden. Über das Symbol  (*show specification*) wird die spezifizierte Menge der vorgegebenen Standorte durch Markierung (dunkelblau) am Bildschirm angezeigt.

4) Spezifikation der Zielfunktion

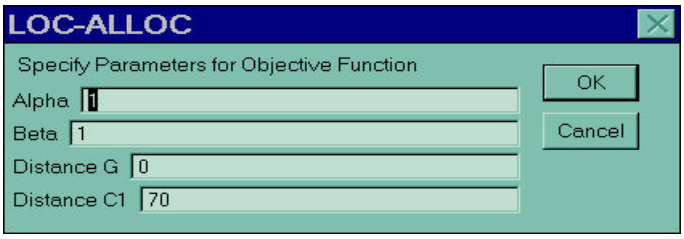
- Unter dem Menüpunkt `Location Criteria` (vgl. Abb. 4.4) kann mit der Option `Objectives` eine andere Spezifikation der Zielfunktion ausgewählt werden. Für dieses Beispiel soll die Minimierung der Zielfunktion Z_3 (*Minimize weighted distance*) gewählt werden (vgl. Abb. 4.16a).

Abb. 4.16a: Menü zur Spezifikation der Zielfunktion



Da für die Spezifikation der Zielfunktion Z_3 noch Parameter notwendig sind (vgl. Kapitel 2, S.11 und S.17), wird ein weiteres Dialogfenster eingeblendet, in dem die Werte für *a* (*Alpha)*, *b* (*Beta*) und *G* eingegeben werden können (vgl. Abb. 4.16b). Für unser Beispiel soll *a*=1, *b*=1 und G=0 sein.

Abb. 4.16b: Menü zur Parameterspezifikation der Zielfunktion Z_3

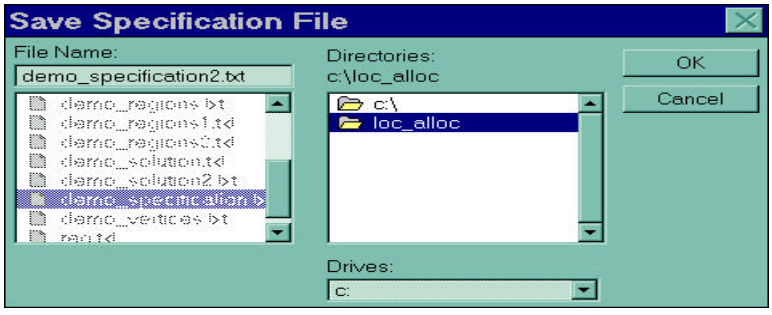


5) Erzeugung einer Parameterdatei für das spezifizierte Standortallokationsproblem

Um für das in den Punkten 1-4 spezifizierte Standortallokationsproblem eine Lösung berechnen zu können, muß die gewählte Spezifikation in einer Datei abgespeichert werden (`Parameterdatei`). Nur Spezifikationen, die in einer Datei enthalten sind, können zur Berechnung herangezogen werden.

- Unter dem Menüpunkt `Location Criteria` mit der Option `Save Specification` werden Sie aufgefordert, einen Dateinamen für die `Parameterdatei` einzugeben (vgl. Abb. 4.17). Geben Sie `demo_specification2.txt` ein und bestätigen Sie mit `OK`.

Abb. 4.17: Erzeugen der `Parameterdatei`



Haben Sie die `Parameterdatei` erzeugt, setzen Sie beim Punkt 4 der Übung 1 (Modellrechnung zur Lösung des gegebenen Standortallokationsproblems, vgl. Abb. 4.7,

S.37) fort. In Abb. 4.18 und Abb. 4.19 sind die Ergebnisse der Modellrechnung dargestellt.

Abb. 4.18: Angebotsstandorte (supply locations) und Zuordnungen der Nachfragestandorte zu den Angebotsstandorten (service areas): Übungsbeispiel 2

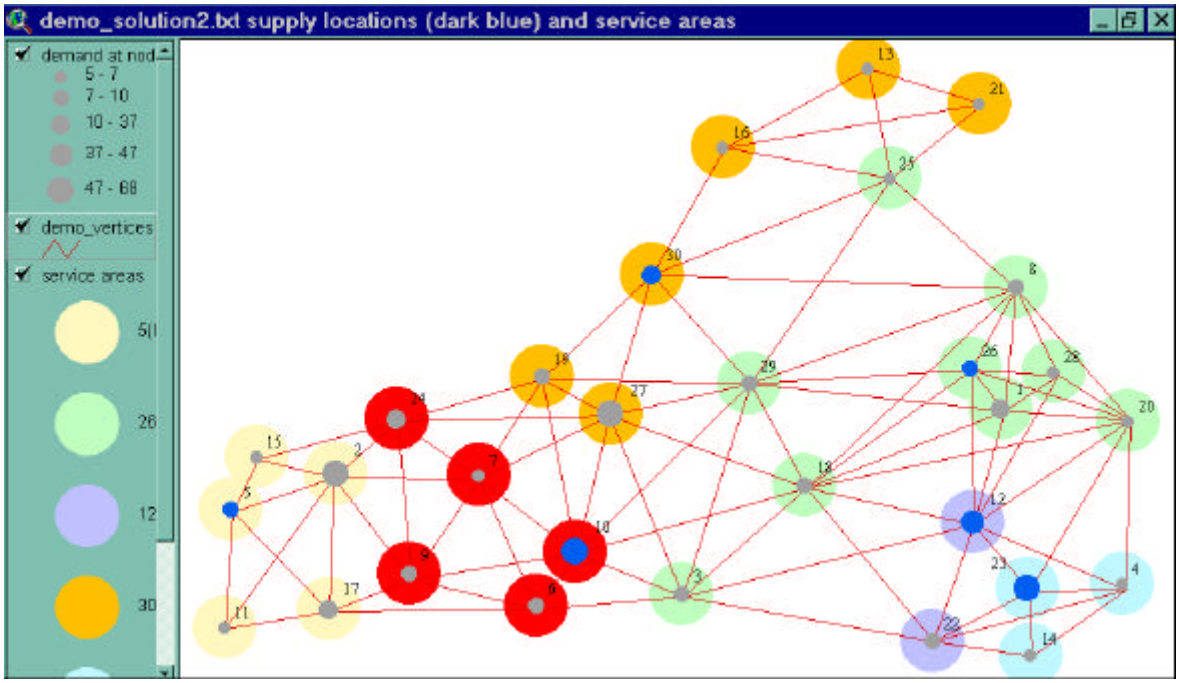
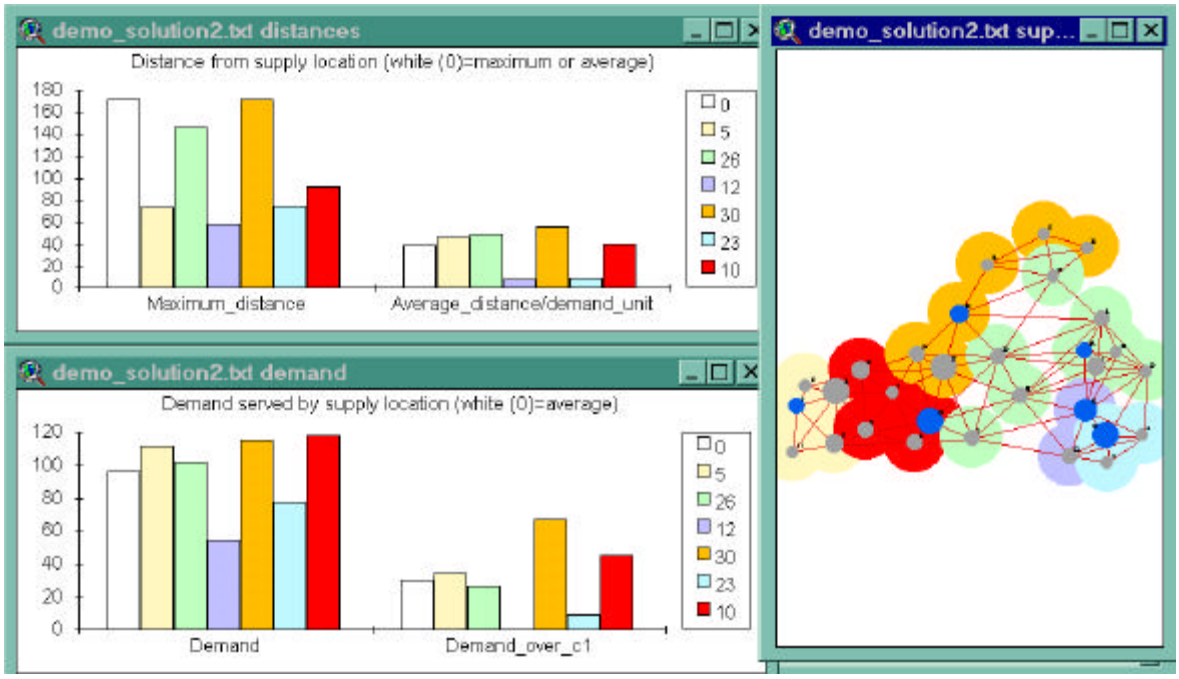


Abb. 4.19: Darstellung der Zielfunktionswerte und der Nachfragewerte für die sechs Angebotsstandorte in Form von Balkendiagrammen: Übungsbeispiel 2



4.2 Neueinteilung der Postamtsbezirke des südwestlichen Niederösterreichs: ein realweltliches Anwendungsbeispiel

Die aktuelle Diskussion um Rationalisierungsmaßnahmen bei der Post im Zuge der Auflösung der Monopolstellung und dem Einsteigen von Konkurrenten legt das hypothetische Fallbeispiel nahe. Es geht darum, Postamtsbezirke zusammenzulegen und „neue" Postämter (Angebotsstandorte) für die neu geschaffenen Postamtsbezirke zu finden, welche bestimmte Eigenschaften aufweisen sollen, die als Randbedingungen in die Modellrechnung eingehen.

Problemstellung und Datenlage

Als Untersuchungsraum werden die Postamtsbezirke im südwestlichen Niederösterreich herangezogen (vgl. Abb. 4.20). Die Anzahl der in einem Postamtsbezirk lebenden Personen wird dem jeweiligen Postamt als Nachfrage zugeordnet. Insgesamt befinden sich im Untersuchungsraum **165 Postämter, die zu zehn „neuen" Postamtsbezirken zusammengefaßt** werden sollen. Für jeden „**neuen" Postamtsbezirk soll ein Postamt (Angebotsstandort) bestimmt werden**. Im gesamten Gebiet leben 583.408 Personen. Die Distanzen zwischen den Postämtern werden in Straßenkilometern gemessen (vgl. Abb. 4.21). Die zehn neu zu schaffenden Postamtsbezirke sollen **folgende Eigenschaften** aufweisen, die als **Randbedingungen in die Modellrechnung** eingehen:

Erstens, die Postamtsbezirke sollen das **Untersuchungsgebiet vollständig abdecken**, die Einzugsbereiche sollen sich nicht überschneiden.
Zweitens, die **Nachfragestandorte** dürfen **nicht mehr als 35 km** von dem zugeordneten Postamt (Angebotsstandort) **entfernt** sein.
Drittens, jeder Postamtsbezirk muß **mindestens 30.000 Einwohner** und **darf höchstens 80.000 Einwohner** besitzen.
Viertens, diejenigen Postämter, die bereits zum jetzigen Zeitpunkt **mehr als 15 000 Einwohner** versorgen, werden a priori als **fixe Postämter** in dem System vorgegeben.
Fünftens, Postämter, die **weiter als 3 km von einem Bahnhof entfernt** sind, werden als „neue" **Postämter (Angebotsstandorte) ausgeschlossen** (vgl. Abb. 4.22).

Erstes Planungsziel bei der Schaffung der zehn „neuen“ Postämter (Angebotsstandorte) mit den zugeordneten Postamtsbezirken ist die Erfüllung der fünf geforderten Eigenschaften (Randbedingungen). **Zweites Planungsziel** ist eine Minimierung der Transportkosten in Bezug auf die vorhandene Nachfrage. Es muß also die Zielfunktion Z_3 herangezogen werden, da sie Nachfrage (Bevölkerung) und Transportkosten (Distanz) in Beziehung setzt. Die Werte für Z_2 sollen ebenfalls berechnet werden, mit C_1=10 km. Die Fragestellung lautet also, welcher Anteil der Bevölkerung eine größere Distanz als 10 km vom zugeordneten Angebotsstandort entfernt ist.

Methodisch gesehen kann zur Lösung diese Standortallokationsproblemes eine **Kombination von Modelltyp 3** (Modell zur Maximierung der Erreichbarkeit bei gegebener Anzahl der Angebotsstandorte und zusätzlichen Erreichbarkeitserestriktionen) und **Modelltyp 5** (Modelle mit Kapazitätsrestriktionen) herangezogen werden.

Die in Abb. 4.23 visualisierten „neuen“ Postämter (dunkelblau markiert) und die zugehörigen „neuen“ Postamtsbezirke sind die **Lösung des Standortallokationsproblems**. Die in Abb. 4.24 dargestellten **Diagramme** entsprechen vom Aufbau den in Übung 1: Punkt 6 (vgl. S. 40,41) vorgestellten Diagrammen. Der weiße Balken (in der Legende mit 0 (Null) bezeichnet) gibt wieder die Durchschnittswerte (für die *maximalen Transportkosten* den Maximalwert) für alle Angebotsstandorte an. Über diese Durchschnittswerte werden unterschiedliche Modellrechnungen vergleichbar. Aus dem Diagramm ***Maximale Transportkosten*** wird ersichtlich, daß kein Angebotsstandort von einem seiner zugeordneten Nachfragestandorte mehr als 35 km entfernt ist. Das Diagramm ***Versorgte Nachfrage*** zeigt, daß alle Angebotsstandorte zwischen 30.000 und 80.000 Einwohner versorgen, was den unteren und oberen Kapazitätsbeschränkungen entspricht. Die „neuen“ Postämter (Angebotsstandorte) entsprechen auch den in Abb. 4.22 visualisierten Randbedingungen. Damit kann das erste Planungsziel als erfüllt betrachtet werden. Die Werte für das zweites Planungsziel, eine Minimierung der Transportkosten in Bezug auf die vorhandene Nachfrage, wird im Diagramm ***Durchschnittliche Transportkosten pro Einwohner*** dargestellt. Die Werte für die Zielfunktion Z_2 sind in dem Diagramm **V*ersorgte Nachfrage über 10 km Entfernung*** enthalten.

Abb. 4.20: Untersuchungsraum südwestliches Niederösterreich mit Postämtern

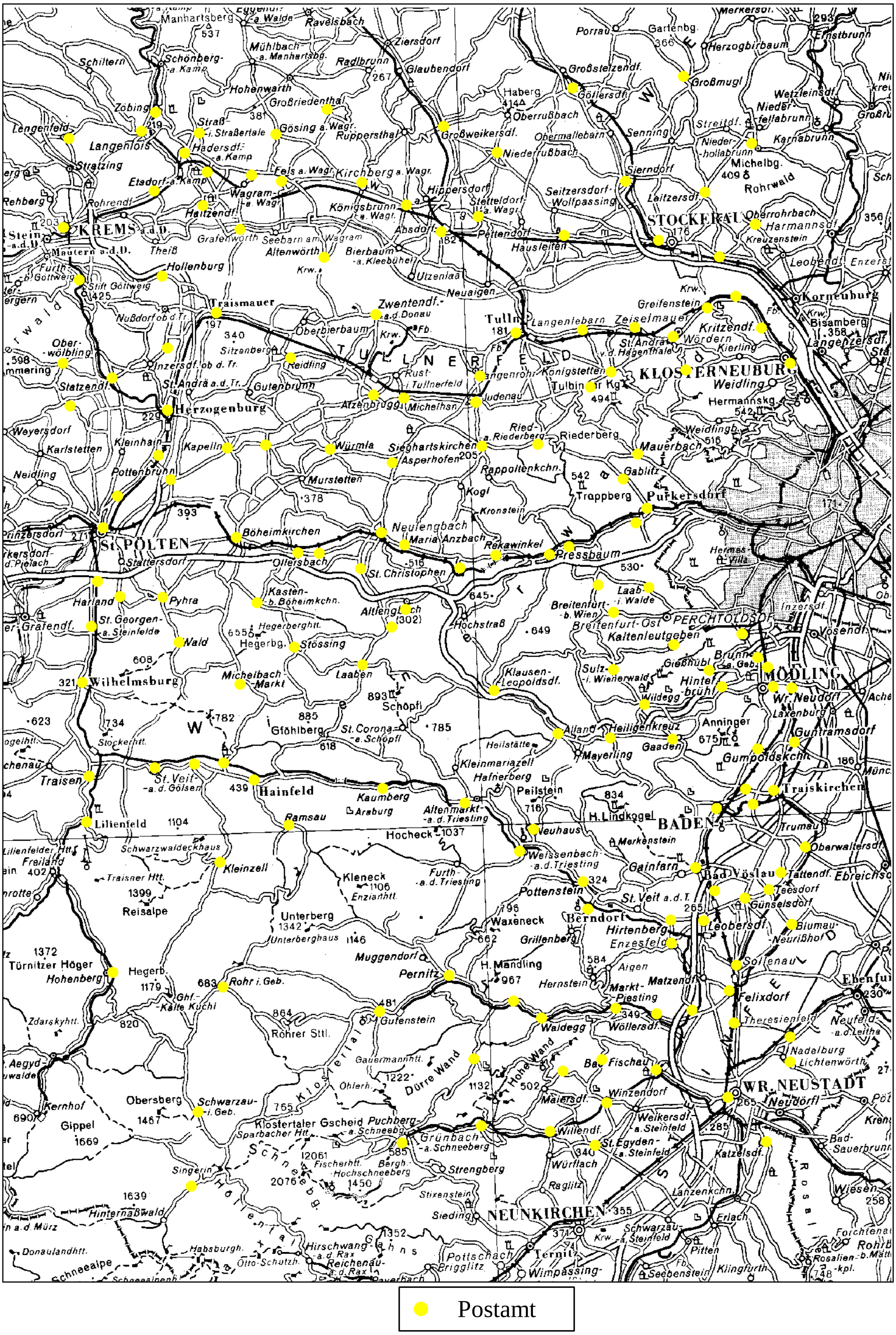


Abb. 4.21: Postämter mit zugeordneter Bevölkerungszahl und Straßenverbindungen

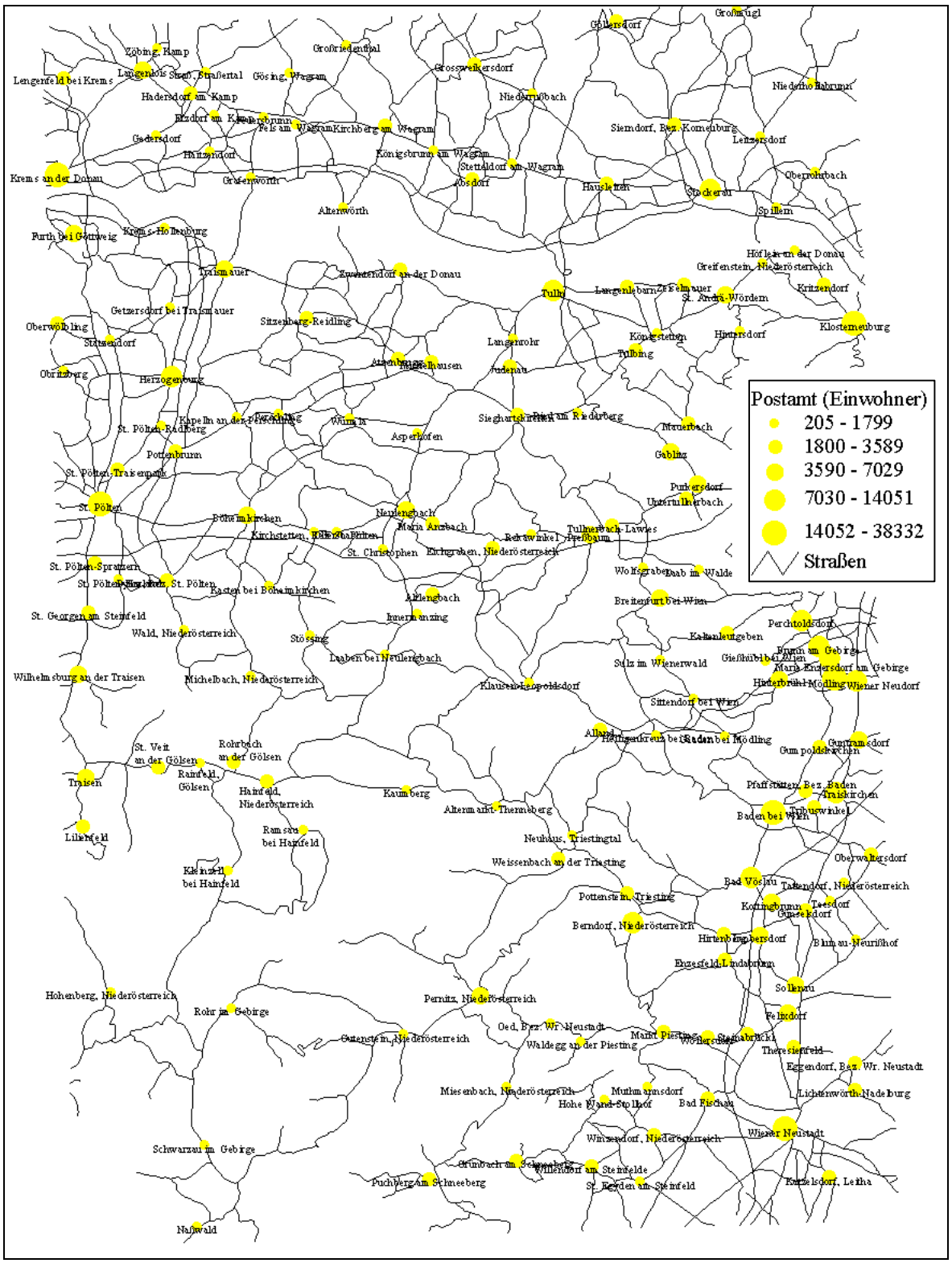


Abb. 4.22: Vorgegebene und ausgeschlossene Postämter

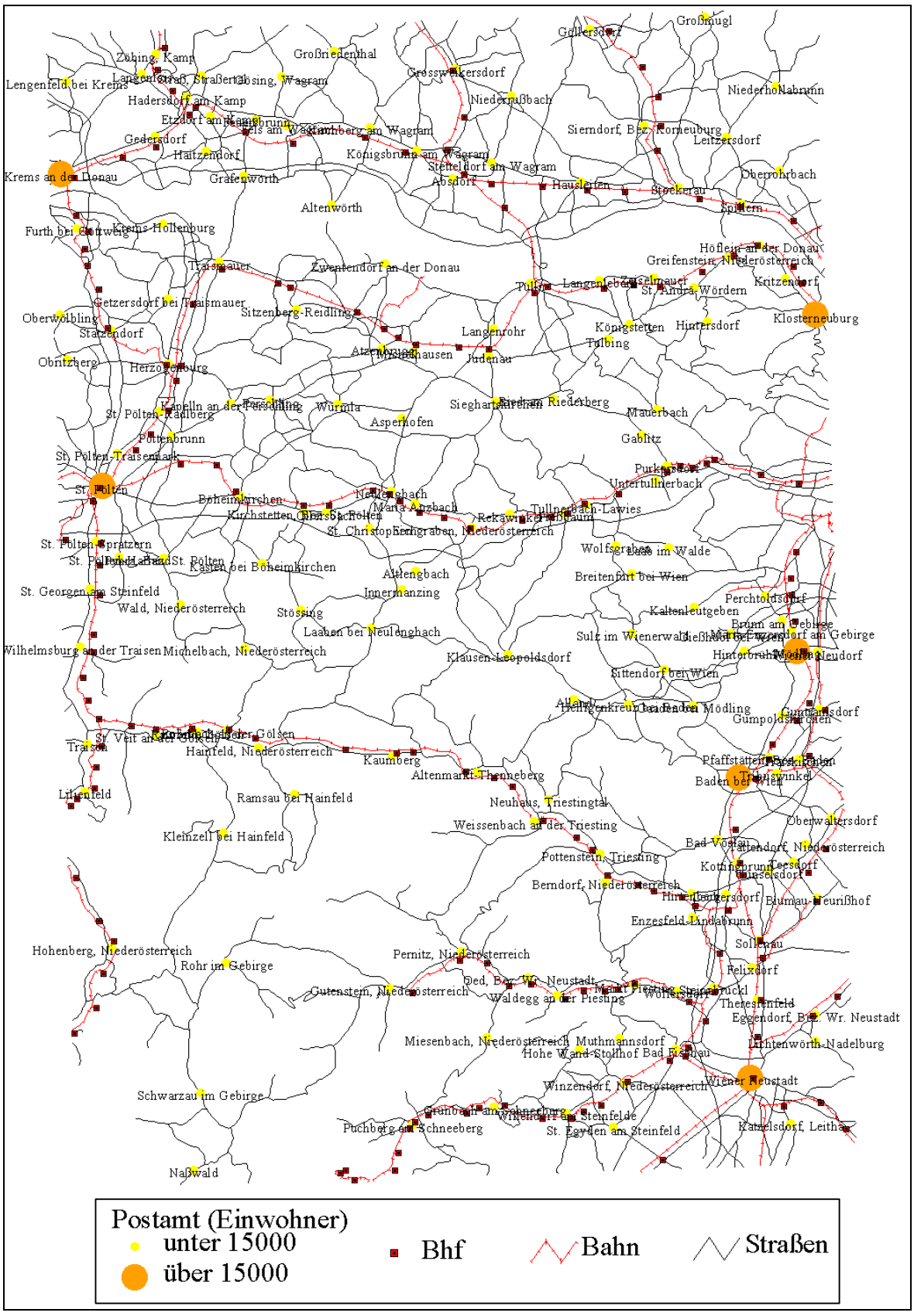


Abb. 4.23: „Neue“ Postamtsbezirke mit „neuen“ Postämtern (Angebotsstandorte)

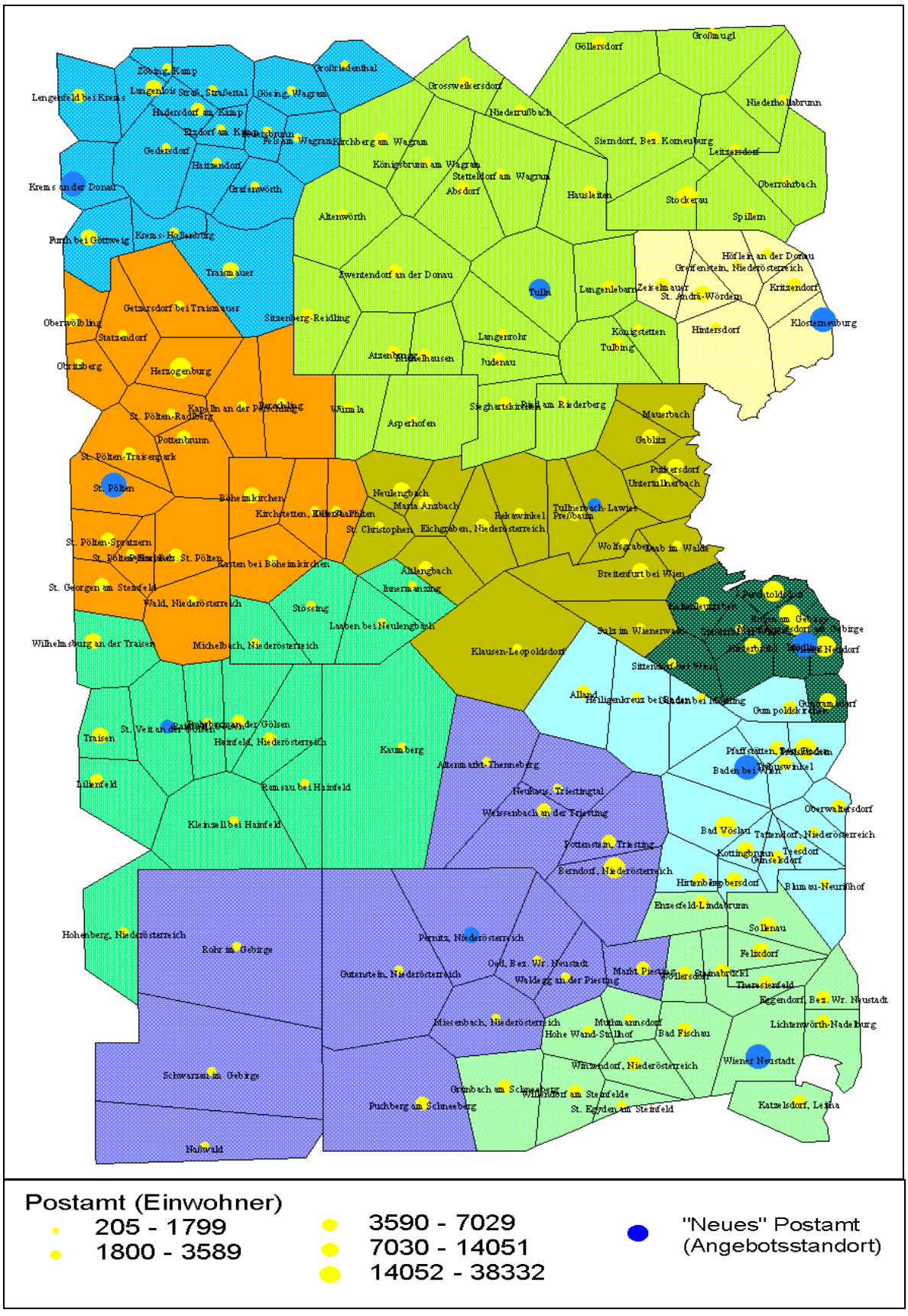
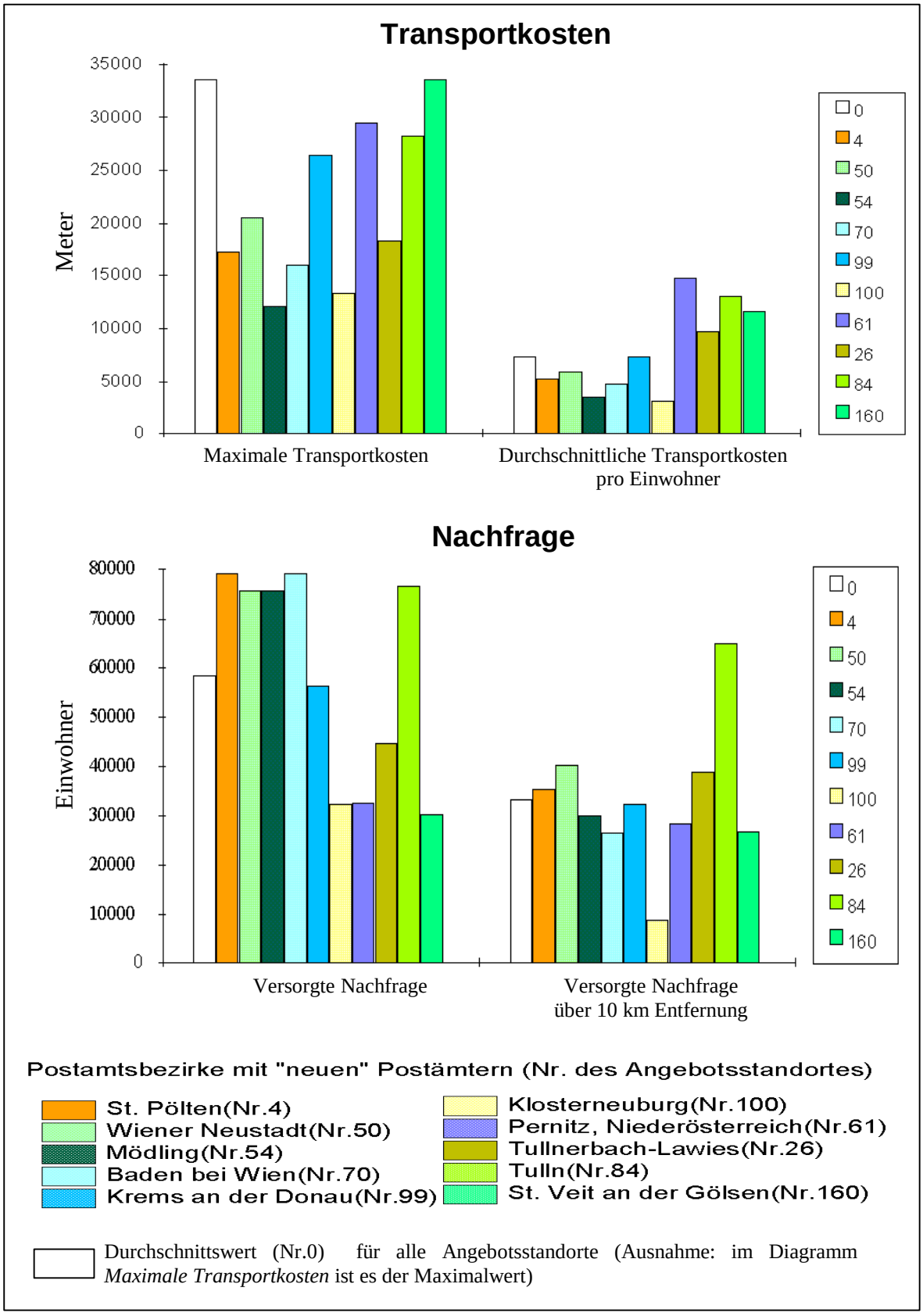


Abb. 4.24: Darstellung der Zielfunktionswerte und der Nachfragewerte für die einzelnen Angebotsstandorte in Form von Balkendiagrammen: „neue“ Postämter



5. Zusammenfassung und Ausblick

Zielsetzung der Arbeit war es, ein **Softwarepaket zur Lösung von statisch-diskreten Standortallokationsproblemen** zu entwickeln, das den Einsatz räumlicher Optimierungsmodelle in der Praxis wesentlich erleichtern soll. Voraussetzung für den Einsatz in der Praxis ist eine **bedienungsfreundliche graphische Benutzeroberfläche**. Ein weiteres Anliegen der vorliegenden Arbeit war es, die **GIS-Technologie** nutzbringend mit der Technologie statisch-diskreter Standortallokationsmodelle im Kontext **realweltlicher Problemstellungen** zu verbinden.

Das in dieser Arbeit beschriebene **Softwarepaket `LOC-ALLOC`** dient zur Lösung von statisch-diskreten Standortallokationsproblemen mit Hilfe von drei elementaren Modelltypen der Klasse der statisch-diskreten Standortallokationsmodelle. Dabei handelt es sich um:

- Modelle zur Maximierung der Erreichbarkeit bei gegebener Anzahl der Angebotsstandorte,
- Modelle zur Maximierung der Erreichbarkeit bei gegebener Anzahl der Angebotsstandorte und zusätzlichen Erreichbarkeitsrestriktionen, und um
- Modelle mit Kapazitätsrestriktionen.

Auch eine Kombination der einzelnen elementaren Modelltypen ist möglich. Das diesem **Modellsystem** zur Lösung von statisch-diskreten Standortallokationsproblemen als Grundlage dienende **heuristische Verfahren** wurde in einem C++ Programm implementiert.

Die entwickelte **graphische Benutzeroberfläche `GUI-LOC-ALLOC`** stellt dem Anwender die folgenden Funktionen zur Verfügung:

- Visualisierung der Problemstellung,
- Spezifikation des Standortallokationsproblems über eine Menüleiste oder durch graphische Auswahl (GIS-Basisfunktionen stehen zur Unterstützung zur Verfügung),
- menügesteuerter Aufruf des Modellsystems,

- Visualisierung der Modellergebnisse in Form einer Karte und in Form von Balkendiagrammen zur Darstellung grundlegender statistischer Kennzahlen für das gefundene Standort-Zuordnungssystem.

Als grundlegender Ansatz für die **Verbindung von Modellsystem und GIS-Technologie** wurde eine **lose Koppelung des Modellsystems mit dem Desktop-GIS ArcView**Ô gewählt. Dieser Ansatz ermöglicht einerseits eine hohe Flexibilität und kann andererseits in beiden Bereichen die jeweiligen Stärken des Systems nutzen (vgl. FISCHER 1996). Der Datenaustausch erfolgt über Textdateien. Im Zuge der Entwicklung dieses Softwarepaketes wurde ein weiterer Ansatz zur Koppelung von GIS und Modellsystem durchgeführt. Es besteht die **Möglichkeit, die graphische Benutzeroberfläche `GUI-LOC-ALLOC` vollständig in das Desktop-GIS ArcView**Ô **3.0 zu integrieren**, und damit die Analysefunktionen von `LOC-ALLOC` innerhalb des GIS in einer einheitlichen Benutzeroberfläche zur Verfügung zu stellen.

Die Lösung einer bestehenden hypothetischen **realweltlichen Problemstellung** sollte die nutzbringende Verbindung von Modellsystem und GIS demonstrieren. Das realweltliche Beispiel bezieht sich auf eine **Neueinteilung der Postamtsbezirke des südwestlichen Niederösterreichs**, etwa um entsprechende Kosten im Kontext der Rationalisierung der Post einsparen zu können. Lösung der Problemstellung und Visualisierung der Ergebnisse zeigen das Potential, das in einer Verbindung von Modellsystem und GIS liegt.

Die in dieser Arbeit behandelten Modelltypen der Klasse statisch-diskreter Standortallokationsmodelle ist nur ein sehr kleiner Ausschnitt aus der Familie der räumlichen Optimierungsmodelle. Die Entwicklung von neuen Verfahren im Bereich **genetischer Algorithmen und Neurocomputing** wird bald eine Erweiterung der Lösungsmöglichkeiten für **andere Problemstellungen der räumlichen Optimierung** mit sich bringen. Die Koppelung der zu entwickelnden Modellsysteme mit einem GIS ist nach Ansicht des Autors eine unerläßliche Forderung für künftige Modellierungen von räumlichen Systemen.

Literaturverzeichnis

BAHRENBERG, G., MATHISSEN, G. und STEINGRUBE, U. (1979): **STAL. Heuristische Algorithmen zur Lösung statisch-diskreter Standort-Allokationsprobleme mit disjunkten Einzugsbereichen**. Materialien und Manuskripte, Heft 2, Universität Bremen, Schwerpunkt Geographie.

BIRKIN, M. (1996): Retail location modelling in GIS, in LONGLEY, P. und BATTY, M. (Hrsg.): **Spatial Modeling in a GIS Environment**, S.207-225. Glasgow: Bell & Bain.

CHURCH, R.L. und SORENSEN, P. (1996): Integrating normative location models into GIS: problems and prospects with the p-median model, in LONGLEY, P. und BATTY, M. (Hrsg.): **Spatial Modeling in a GIS Environment**, S.167-183. Glasgow: Bell & Bain.

DENSHAM, P.J. und RUSHTON, G. (1992): Strategies for solving large location-allocation problems by heuristic methods, **Environment and Planning A** 24, S. 289-304.

DENSHAM, P.J. (1996): Visual interactive locational analysis, in LONGLEY, P. und BATTY, M. (Hrsg.): **Spatial Modeling in a GIS Environment**, S.184-205. Glasgow: Bell & Bain.

ESRI (1990): **Arc/Info User´s Manual**. Redlands CA, Enviromental Systems Research Institut.

ESRI (1996): **ArcView 3.0 User´s Manual**. Redlands CA, Enviromental Systems Research Institut.

ESRI (1996): **ArcView 3.0 Avenue Manual**. Redlands CA, Enviromental Systems Research Institut.

FISCHER, M.M. (1992): **Räumliche Optimierungsmodelle zur Lösung von Standort-Allokationsproblemen**. Theorien, Modelle und Methoden in der Wirtschaftsgeographie, Teil A. Institut für Wirtschafts und Sozialgeographie, Wirtschaftsuniversität Wien.

FISCHER, M.M., SCHOLTEN, H.J. und UNWIN, D. (1996): Geographic information systems, spatial data analysis and spatial modelling: an introduction, in FISCHER, M.M., SCHOLTEN, H.J. und UNWIN, D. (Hrsg.): **Spatial Analytical Perspectives on GIS**, S.3-18. London Taylor & Francis.

LONGLEY, P. und BATTY, M. (1996): Analysis, modelling, forecasting, and GIS technology in LONGLEY, P. und BATTY, M. (Hrsg.): **Spatial Modeling in a GIS Environment**, S. 1-15. Glasgow: Bell & Bain.

MARANZANA, F. (1964): On the location of supply points to minimize transport costs. **Operational Research Quaterly** 15, S. 261-270.

NARULA, S.C. und OGBU, U.I. (1979): An hierarchical location-allocation problem. **Omega,** 7 (Nr.2), S. 137-143.

REVELLE, C.S. und SWAIN, R.W. (1970): Central facilities location, **Geographical Analysis** 2, S. 30-42.

ROSING, K.E. und HARRIS, B. (1992): Algorithmic and technical improvements: optimal solutions to the (Generalized) Multi-Weber problem, **Papers in Regional Science**, 71, S. 331-352.

TEITZ, M.B. und BART, P. (1968): Heuristic methods for estimating the generalized vertex median of a weithed graph. **Operational Research** 16, S. 955-961.

Anhang A: Installation und Bedienung von LOC-ALLOC

In A.1 geht es um den **Anwendungsbereich** und die Softwarevoraussetzungen des Programmpaketes. In A.2 wird die **Installation** des Programmes, der graphischen Benutzerschnittstelle und der Beispieldaten beschrieben, während A.3 sich mit der Struktur der zu verwendenden **Eingabedateien** beschäftigt. Der **direkte Aufruf des Programmes** ohne Verwendung der graphischen Benutzerschnittstelle sowie der Inhalt der erzeugten Ergebnisdatei wird in A.4 erklärt. Ist es, aus welchen Gründen auch immer, nicht möglich mit der graphischen Oberfläche zu arbeiten, so sind A.3 und A.4 durchzuarbeiten, nachdem der in A.2 beschriebene Installationsvorgang abgeschlossen wurde. Sollten schon Programm und graphische Benutzerschnittstelle installiert sein, so empfiehlt sich der Einstieg mittels der in 4. beschriebenen Übungsbeispiele. Über sie erhält man einen guten Einblick in die Funktionsweise von LOC-ALLOC und kann schon bald eigenständig Aufgaben lösen. Weiterführende Arbeiten oder Unklarheiten bei den Übungsbeispielen können durch die in A.5 gegebenen **Menü- und Funktionsbeschreibungen** gelöst werden. Mit dem Programmpaket werden die Dateien für ein Demobeispiel und ein Anwendungsbeispiel mitgeliefert.

A.1 Anwendungsbereich

LOC-ALLOC ist ein **Programmpaket zur Lösung von elementaren Standort-Allokationsproblemen mit Hilfe von Modellen des Typs 2, 3 und 5** oder einer **Kombination** dieser (vgl. Kap. 2.3). Es läuft unter unter Microsoft Windows 3.1, Windows 95 und Windows NT. Das Programm des Algorithmus ist in C++ (Ansi-Standard) implementiert und arbeitet ausschließlich mit Textdateien zur Ein- und Ausgabe. Damit ist es möglich, das Programm unter jedem Betriebssystem anzuwenden, unter dem ein C++ Compiler verfügbar ist.

Die graphische Benutzerschnittstelle ist in der Makrosprache AVENUE™ des Desktop-GIS ArcView™ 3.0 programmiert. Um sie zu benutzen, ist es notwendig, eine Version von ArcView™ 3.0 lizensiert zu haben. Da ArcView™ Versionen für Windows, Unix und Apple existieren, kann man auch unter diesen Betriebssystemen mit der graphischen Oberfläche von LOC-ALLOC arbeiten.

A.2 Installation

Wird das Programmpaket unter einem Betriebssystem installiert, das nur 8 Zeichen lange Dateinamen kennt, so sind die hier angegebenen Namen dementsprechend zu kürzen. Ein Verzeichnis mit dem Namen **c:\loc_alloc** ist zu erstellen, in das alle Programmdateien von LOC-ALLOC und die Beispieldateien kopiert werden müssen!

C++ Programm:
Das kompilierte C++ Programm **loc_alloc.exe** wird in das Verzeichnis **loc_alloc** kopiert. Ist für das am Rechner installierte Betriebssystem noch keine ausführbare Version vorhanden, so muß der Source Code aus der Datei loc_alloc.cpp erst auf einem für das Betriebssystem geeigneten C++ Compiler compiliert werden.

Graphische Benutzerschnittstelle:
Sie liegt in Form eines ArcView™ Projektes (Dateiname: gui_loc_alloc.apr) vor und muß in das selbe Verzeichnis wie das Programm loc_alloc.exe kopiert werden.

Beispieldaten:
Das Softwarepaket enthält ein Übungsbeispiel **demo** und das Anwendungsbeispiel **post**. Für das Übungsbeispiel sind die in Tab. A.1 angegebenen 15 Dateien zu kopieren und für das Anwendungsbeispiel werden die in Tab. A.2 angegebenen 15 Dateien benötigt.
Wichtig: Die Dateien der Beispiele müssen ins Verzeichnis c:\loc_alloc kopiert werden!

Tab. A.1: Dateien des Übungsbeispieles demo

Textdateien	demo_nodes.txt	demo_distances.txt
	demo_vertices.txt	demo_specification.txt
Graphikdateien		
demo_nodes.shp	demo_vertices.shp	demo_service_areas.shp
demo_nodes.shx	demo_vertices.shx	demo_service_areas.shx
demo_nodes.dbf	demo_vertices.dbf	demo_service_areas.dbf
demo_nodes.avl	demo_vertices.avl	demo_service_areas.avl

Tab. A.2: Dateien des Anwendungsbeispiels `post`

Textdateien	post_nodes.txt	post_distances.txt
	post_vertices.txt	post_specification.txt
Graphikdateien		
post_nodes.shp	post_vertices.shp	post_service_areas.shp
post_nodes.shx	post_vertices.shx	post_service_areas.shx
post_nodes.dbf	post_vertices.dbf	post_service_areas.dbf
post_nodes.avl	post_vertices.avl	post_service_areas.avl

A.3 Eingabedateien

Das Programm arbeitet mit 3 **Textdateien**, die die **Definition des Graphen** enthalten auf den das Standortallokationsmodell angewendet werden soll. In einer weiteren Textdatei sind die Parameter enthalten, die ein spezifisches Standortallokationsproblem definieren. Zur Definition des Graphen ist die Knotendatei (Abb. A.1) notwendig, die die Nummern (*node_nr*) der Knoten und die jeweils zu versorgende Nachfrage (*demand*) enthält. In einer zweiten Datei (Abb. A.2) sind die Kanten des Graphen enthalten.

Abb. A.1: Beispiel einer Knotendatei (`demo_nodes.txt`)

```
node_nr,demand
11,6
17,35
...,...
```

Abb. A.2: Beispiel einer Kantendatei (`demo_vertices.txt`)

```
from,to
11,5
11,2
15,2
...,...
```

Zur Lösung des Standortallokationsproblemes ist es notwendig die kürzesten Distanzen für alle Verbindungen innerhalb des Graphen zu berechnen. Sie sind in der dritten Datei (Abb. A.3) enthalten.

Abb. A.3: Beispiel einer Kürzeste Distanzendatei (`demo_distances.txt`)

```
from,to,distance
1,2,260
1,3,140
1,4,138
...,...
30,29,83
```

Die **Parameterdatei** dient zur **Spezifikation der Randbedingungen und der Zielfunktion** (Tab. A.3a). Variablenname und Wert sind durch ein "=" zu trennen; und die Variable *number_of_supply_locations* hat vor den anderen Variablen der Randbedingungen zu stehen. Bestimmte Variablen sind obligatorisch (z.B.: *number_of_supply_locations =6*), andere fakultativ (z.B.: *min_capacity=540*). Mit *n* ist in den folgenden Tab. A.3a und A.3b die Anzahl der Knoten bezeichnet.

Für die Durchführung des heuristischen Verfahrens sind weitere 3 Variablen als **Parameter für den Algorithmus** notwendig (vgl. Tab. A.3b). Da der Erfolg dieses Verfahrens von der Startkonfiguration abhängig ist (vgl. Kap. 3.2), gibt es die Möglichkeit, diese Konfiguration der Angebotsstandorte direkt vorzugeben (*start;* vgl. Tab. A.3b). Soll das nicht getan werden, so besteht die Möglichkeit, daß das Verfahren mit einer unterschiedlichen zufällig gewählten Startkonfiguration mehrere Male durchgeführt wird, und der beste Wert für die zu optimierende Zielfunktion wird gemeinsam mit der Startkonfiguration in der Parameterdatei abgespeichert. Wie oft das Verfahren mit einer zufällig gewählten Startkonfiguration durchgeführt wird, ist mit den Parametern *trial_solution* und *trial_optimize* anzugeben. *trial_solution* bezieht sich dabei auf das *Teilverfahren zum Auffinden eines Standort-Zuordnungssystems, das den geforderten Randbedingungen (ohne* $\boldsymbol{R}_4$ *und* $\boldsymbol{R}_5$*) entspricht* (vgl. Kap. 3.2, S.23) und *trial_optimize* bezieht sich auf das *Teilverfahren zur Optimierung des gefundenen Standort-Zuordnungssystems* (vgl. Kap. 3.2, S.26). Durch diese Parameter besteht die Möglichkeit, das heuristische Verfahren automatisch mit einer Vielzahl von Startkonfigurationen durchführen zu lassen.

Tab. A.3a: Variablennamen und Wertbereiche der `Parameterdatei`

Variabelnamen	Bedeutung	Wertebereich	obligatorisch
Randbedingungen			
number_of_ supply_locations	(R_6) Anzahl der Angebotsstandorte	ganze Zahlen [1, ..., n]	ja
fixed	(R_9) vorgegebene Standorte ($\boldsymbol{S_1}$)	ganze Zahlen [1, ..., n] durch Komma (",") getrennt	nein
excluded	(R_{10}) ausgeschlossene Standorte ($\boldsymbol{S_0}$)	vgl. oben	nein
min_capacity	(R_7) Kapazitätsuntergrenze ($\boldsymbol{K_u}$)	Dezimalzahlen	nein
max_capacity	(R_8) Kapazitätsobergrenze ($\boldsymbol{K_o}$)	Dezimalzahlen	nein
max_distance	(R_4) Maximale Distanz ($\boldsymbol{C_o}$)	Dezimalzahlen	nein
c1	(R_5) Zweite Distanz ($\boldsymbol{C_1}$)	Dezimalzahlen	nein
a1	(R_5) Maximales Gewicht ($\boldsymbol{a_1}$) über $\boldsymbol{C_1}$	Dezimalzahlen	nein
Zielfunktion			
Objective	Angabe welche der drei Zielfunktionen minimiert werden soll, abhängig vom angegebenen Wert	"1" für Z_1 "2" für Z_2 "3" für Z_3	ja
alpha	(Z_3) Exponent für Bedarf (a)	ganze Zahlen	nein
beta	(Z_3) Exponent für Distanz (b)	ganze Zahlen	nein
G	(Z_3) Distanz oberhalb der optimiert wird ($\boldsymbol{G}$)	Dezimalzahlen	nein

Tab. A.3b: Variablennamen und Wertbereiche der `Parameterdatei`

Variabelnamen	Bedeutung	Wertebereich	obligatorisch
Parameter für Algorithmus			
trial_optimize	Anzahl der Startkonfigurationen für Optimierung	ganze Zahlen	ja
trial_solution	Anzahl der Startkonfigurationen um eine Lösung zu finden	ganze Zahlen	ja
Start	Standorte mit denen der Algorithmus gestartet wird (Startkonfiguration)	ganze Zahlen [1, ..., n] durch Komma (",") getrennt	nein

Ein Beispiel (Abb. A.4) soll ein mögliches konkretes Aussehen der Parameterdatei bei Programmaufruf ohne Graphikoberfläche veranschaulichen.

Abb. A.4: Beispiel einer `Parameterdatei` für Aufruf ohne Graphikoberfläche

```
number_of_supply_locations =6
fixed=5,26
objective=1
trial_optimize=1
trial_solution=1
start=22,25,1,17
```

Wird das Programm mit der graphischen Benutzerschnittstelle verwendet, so ist es notwendig, die **Namen der Textdateien und die Namen der Graphikdateien** (GIS-Daten) in die Parameterdatei aufzunehmen (vgl. Tab. A.4), da beim automatischen Aufruf von `loc_alloc.exe` über die graphische Benutzerschnittstelle `GUI-LOC-ALLOC` die Namen der Textdateien an das Programm `loc_alloc.exe` übergeben werden müssen. Die Namen der Graphikdateien sind notwendig, um die Visualisierung des Graphen und der Ergebnisse zu ermöglichen. Der Wertebereich für Dateinamen wird als Zeichenkette (String) angegeben, d.h. er besteht aus ASCII Zeichen. Bei den **Graphikdateien** handelt es sich um ArcView™ `Shapefiles` und `Legendfiles` (vgl. ESRI 1996). Das die Knoten des Graphen repräsentierende ArcView™ `Shapefile(`z.B.: `demo_nodes.shp`) ist über die Knotennummer (*node_nr*) mit der Definition des Graphen in den Textdateien `(demo_nodes.txt)` zu verbinden.

Tab. A.4: Variablen für Programmaufruf mit Graphikoberfläche

Variabelnamen	Bedeutung	Wertebereich	obligatorisch
Textdateien			
nodes	Knotendateiname	Zeichenkette (String)	ja
vertices	Kantendateiname	Zeichenkette (String)	ja
distances	Name für Datei mit kürzesten Distanzen	Zeichenkette (String)	ja
Graphikdateien			
nodes_shp	Name von Arc-View Shapefile für Knoten	Zeichenkette (String)	ja
vertices_shp	Name von Arc-View Shapefile für Kanten	Zeichenkette (String)	ja
service_areas_shp	Name von Arc-View Shapefile für Einzugsbereiche	Zeichenkette (String)	ja
demand_leg	Name von Arc-View Legendfile für Knoten	Zeichenkette (String)	ja
service_areas_leg	Name von Arc-View Legendfile für Einzugsbereiche	Zeichenkette (String)	ja

Da bei der Arbeit mit der Graphikoberfläche die Variablenwerte interaktiv über Menüpunkte eingegeben werden, hat der Benutzer nur einen Dateinamen anzugeben und die Parameterdatei wird in der in Abb. A.5 gezeigten Form abgespeichert.

Abb. A.5: `Parameterdatei` für Aufruf mit Graphikoberfläche (`demo_specification.txt`)

```
********** graph files **********
nodes= demo_nodes.txt
vertices= demo_vertices.txt
distances= demo_distances.txt
********** constraints **********
number_of_supply_locations =6
fixed=5,26
********** objective function **********
objective=1
alpha=1
beta=1
G=0
********** algorithm parameters **********
trial_optimize=1
trial_solution=1
start=22,25,1,17
********** graph  arc view representation **********
nodes_shp=c:\loc_alloc\demo_nodes.shp
vertices_shp=c:\ loc_alloc \ demo_vertices.shp
service_areas_shp=c:\ loc_alloc \ demo_service_areas.shp
demand_leg=c:\ loc_alloc \ demo_nodes.avl
service_areas_leg=c:\ loc_alloc \ demo_service_areas.avl
```

A.4 Direkter Programmaufruf und Ergebnisse

Das Programm wird mit dem Befehl **loc_alloc** aufgerufen. Als Parameter werden fünf Dateinamen benötigt. Dies sind die Namen für die Knotendatei, Kantendatei, Parameterdatei, Ergebnisdatei und die kürzeste Wege Datei.

```
c:\loc_alloc\loc_alloc demo_nodes.txt demo_vertices.txt
demo_specification.txt demo_solution.txt demo_distances.txt
```

In der Ergebnisdatei (**demo_solution.txt** in Tab. A.4) werden die berechneten Angebotsstandorte und ihre Zuordnungen abgespeichert. Jeder Einzugsbereich (service area) erhält eine Nummer, von 1 beginnend bis zur Anzahl der zu bestimmenden Angebotsstandorte(*number_of_supply_locations*). Für jeden Knoten (Nachfragestandort) ist angegeben, welchem Angebotsstandort er zugeordnet ist und wieviel er zu den Zielfunktionswerten dieses Angebotsstandortes beiträgt. Bei den Nachfragestandorten, die Angebotsstandorte sind, ist die versorgte Nachfrage und die Zielfunktionswerte für diesen Angebotsstandort mit seinen zugeordneten Nachfragestandorten gespeichert. Im mit 0 (Null) bezeichneten Einzugsbereich sind die Zielfunktionswerte für das Gesamtsystem enthalten.

Tab. A.5: Beispiel für eine Ergebnisdatei

s_area	supply_nr	node_nr	is_supply	node_count	demand	obj_1	obj_2	obj_3
0	0	0	1	30	580	172	80	22753
1	5	2	0	0	40	37	0	1480
...	...							
1	5	5	1	4	114	74	30	5209

Bedeutung der Variablen innerhalb einer Ergebnisdatei

s_area: Nummer des Einzugsbereichs (service area)

supply_nr: Nummer des Angebotsstandorts

node_nr: Nummer des Knotens

is_supply: Ob Knoten ein Angebotsstandorts ist (0: kein Angebotsstandort; 1: ist ein Angebotsstandort)

node_count: Anzahl der Knoten innerhalb eines Einzugsbereichs (service area)

demand: Im Fall **is_cent**=1 enthält dieser Tabelleneintrag die vom Angebotsstandort versorgte Nachfrage; Im Fall **is_cent**=0 entspricht er den zu versorgenden Nachfrage am Standort **node_nr**

obj_1: Wert für Z_1

obj_2: Wert für Z_2

obj_3: Wert für Z_3

In der Parameterdatei werden die **Zielfunktionswerte** für das Gesamtsystem und die Standorte der Startkonfiguration für diese Lösung gespeichert. Dies ist für die Nachvollziehbarkeit von Modellrechnungen notwendig. Die Speicherung erfolgt so, daß die Ergebnisse immer an die bestehende Parameterdatei angehängt werden. Damit sind alle Modellrechnungen mit verschiedenen Startsystemen in der zugehörigen Parameterdatei abgespeichert und es ist möglich, die Ergebnisse mit unterschiedlichen Startkonfigurationen miteinander zu vergleichen. Sind innerhalb des Programmes Fehler aufgetreten, so werden sie in der Parameterdatei mit folgender Syntax "error= *Fehlermeldung* " gespeichert. In diesem Fall wird keine Ergebnisdatei erzeugt.

A.5 Graphische Benutzerschnittstelle

Der **Aufruf** der graphischen Benutzerschnittstelle erfolgt über das Programm ArcView™ (vgl. Kap. 4.1). Die graphische Benutzerschnittstelle erscheint am Bildschirm (vgl. Abb. A.6). Sie besteht aus der **Menüleiste** und aus der **Werkzeugleiste.**

Abb. A.6: Graphische Benutzerschnittstelle (GUI-LOC-ALLOC)

Menüleiste

Über die **Menüleiste** (vgl. Abb. A.7) können sämtliche Funktionen des Systems angesprochen werden. Es folgt eine Beschreibung der einzelnen Menüpunkte und ihrer Optionen. Werden bei den Optionen Dialogfenster aufgerufen, so ist ihre Bedeutung und die erforderliche Eingabe durch den Benutzer beschrieben.

Abb. A.7: Menüleiste

Dateien, die die Daten des Graphen und seiner graphischen Repräsentation enthalten, werden angegeben. Eine Option dient zur Angabe der Textdateien (Graph), die zweite (Graph Representation**)** zur Angabe der Graphikdateien.

- **`Graph:`** Angabe der Dateinamen, die Knotendaten, Kantendaten und kürzeste Distanzen enthalten.
- **`Graph Representation:`** ArcView™ GIS-Dateien der Knoten, Kanten und Einzugsbereiche werden geladen und dargestellt. Weiters müssen die Namen der Legendendatein angegeben werden, mit denen die Nachfrage, die Kanten und die Einzugsbereiche darzustellen sind.

Hier werden unter verschiedenen Optionen die für die Definition eines spezifischen Standortallokationsproblemes notwendigen Parameter eingegeben und bearbeitet (vgl. Abb. 4.4a).

- **`Initialise`**: sämtliche Parameter werden auf ihre Ausgangswerte zurückgesetzt,

- **`Constraints:`** Die Spezifikation der Randbedingungen kann in einem Dialogfeld eingegeben werden (vgl. Kap. 4.1 Übung 2 Punkt 1):
  Number of Supply Locations: Anzahl der Angebotsstandorte

Fixed Locations:	vorgegebene Standorte (können auch direkt über graphische Darstellung und Werkzeugleiste eingegeben werden).
Excluded Locations:	ausgeschlossene Standorte (können auch direkt über graphische Darstellung und Werkzeugleiste eingegeben werden).
Maximum Distance (C_0):	Maximale Distanz (C_0)
Maximum Capacity:	Kapazitätsobergrenze
Minimum Capacity:	Kapazitätsuntergrenze
Distance (C_1):	Distanz C_1
Maximum Demand over C_1 (a_1):	Maximaler Bedarf über C_1 (a_1)

- **Objectives**: Folgende Zielfunktionen können optional gewählt werden (vgl. Kap. 4.1 Übung 2 Punkt 4).

Minimize distance:	Minimiert die Distanzen im Gesamtsystem
Minimize maximum distance:	Minimiert die maximale Distanz im Gesamtsystem
Minimize demand over C_1:	Minimiert den Bedarf oberhalb der Distanz C_1
Minimize weighted distance (*P-Median Problem)*:	Minimiert die gewichtete Distanz (entspricht der Zielfunktion für das P-Median Problem). Für die Parameter für Z_3 folgt ein Dialogfeld.
alpha:	Exponent *a*
beta:	Exponent *b*
G:	Distanz *G*

- **Algorithm Parameters:** Parameter für den Berechnungsalgorithmus

Starting Locations:	Startsystem mit dem der Algorithmus seine Berechnung starten soll (kann auch direkt über graphische Darstellung und Werkzeugleiste eingegeben werden).
Trial Solution:	Anzahl der Versuche mit einer zufälligen Startkonfiguration um eine Lösung des Systems zu finden.
Trial Optimization:	Anzahl der Versuche mit einer zufälligen Startkonfiguration um die Lösung weiter zu optimieren

- **Save Specification:** Die in den vorherigen Menüpunkten eingegebenen Parameter werden in einer Parameterdatei im Textformat abgespeichert und diese Datei wird zur aktuellen für den nächsten Aufruf von **solve**. Hier sind alle für die Berechnung eines spezifischen Standortallokationsproblemes notwendigen Daten enthalten (vgl. Kap. 4.1 Übung 2 Punkt 5).

- **Load Specification:** Eine schon existierende Parameterdatei kann geladen werden. Ihre graphische Repräsentation wird dargestellt, sofern die Dateinamen in der Parameterdatei angegeben sind (vgl. Kap. 4.1 Übung 1 Punkt 3).

Solve

- **Start Computation:** Das in der aktuellen Parameterdatei (die beispielsweise zuvor mit **Save Specification** erstellt wurde oder mit **Load Specification** geladen wurde) spezifizierte Standortallokationsproblem wird gelöst. Die für die Berechnung zu verwendenden Dateinamen und Parameter werden nochmals angezeigt und nach der Bestätigung wird der Benutzer aufgefordert einen Namen für die Ergebnisdatei anzugeben. In der Ergebnisdatei sind sowohl die Zuordnungen der Nachfragestandorte zu den jeweiligen Angebotsstandorten enthalten, als auch die Zielfunktionswerte für Nachfragestandorte, Einzugsgebiete und das Gesamtsystem. Konnte das Standortallokationsproblem nicht gelöst werden, so wird in einem File die aufgetretene Fehlermeldung abgespeichert und angezeigt (vgl. Kap. 4.1 Übung 1 Punkt 4).

Display Solution

Die Ergebnisse der Modellrechnung werden visualisiert. Ergebnisdateien von schon durchgeführten Modellrechnungen können geladen werden (vgl. Abb. 4.10).

.

- **Select Solution File:** Hier kann ein Ergebnisdateiename angegeben werden um eine schon erzeugte Ergebnisdatei zu laden.
- **Display Supply Locations and Service Areas:** Visualisiert die Angebotsstandorte (supply locations) und Zuordnungen der Nachfragestandorte

zu den Angebotsstandorten (service areas) für die aktuelle Ergebnisdatei auf der graphischen Repräsentation des Graphen (vgl. Abb. 4.11).

- **Display Charts for Distances and Demand:** Darstellung der Zielfunktionswerte und der Nachfragewerte für die einzelnen Angebotsstandorte in Form von Balkendiagrammen für die aktuelle Ergebnisdatei (vgl. Abb. 4.12).

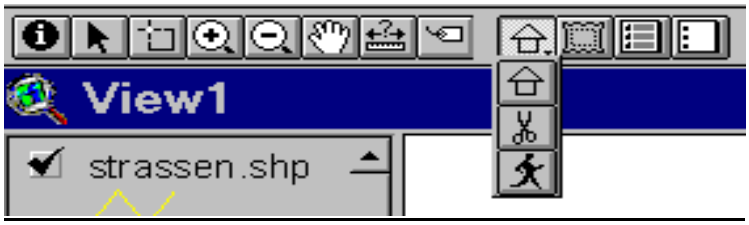
- **Clear Display for New Computation:** Löscht die Darstellung der Angebotsstandorte mit ihren Zuordnungen und die Balkendiagramme aus dem Arbeitsspeicher Anzuwenden, bevor weitere Berechnungen durchgeführt werden (vgl. Kap. 4.1 Übung 1 Punkt 7).

- **Change Interface:** Schaltet zwischen Benutzeroberfläche mit ArcView und solcher ohne ArcView Funktionen um

Werkzeugleiste

Die **Werkzeugleiste** (Abb. A.8) dient einerseits dazu Daten abzufragen, über die graphische Darstellung des Graphen. Andererseits ist es möglich, über die graphische Darstellung Randbedingungen zu spezifizieren.

Abb. A.8: Werkzeugleiste



Funktionen zur Arbeit mit der graphischen Darstellung, und zur Auswahl von Objekten mit Hilfe der graphischen Darstellung

Information Die vorhandenen Informationen über das gewählte Objekt werden in einem Fenster dargestellt,

Auswahl-Rechteck Der Cursor erscheint als Rechteck. Mit Hilfe dieses Rechtecks können Standorte ausgewählt werden, die dann dunkelblau erscheinen (vgl. Abb. 4.14),

Vergrösserung Der gewählte Bildausschnitt wird vergrössert (Zoom),

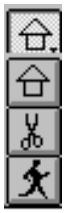
Verkleinerung Der gewählte Bildausschnitt wird verkleinert (Pan),

Bild ziehen Der Bildausschnitt wird verschoben,

Messen Abstände (Distanzen) innerhalb des Graphen können gemessen werden,

Benennen Fügt Namen des Objektes ein.

Funktionen zur graphischen Spezifikation von Randbedingungen



Über das Werkzeugmenü kann ausgewählt werden auf welche Menge (*set*) von Nachfragestandorten die folgenden Operationen angewendet werden (Menge der vorgegebene Angebotsstandorte, Menge der ausgeschlossene Angebotsstandorte oder Menge der Nachfragestandorte der Startkonfiguration). Durch anklicken und festhalten der linken Maustaste werden alle Symbole sichtbar. Es besteht die Möglichkeit ein Symbol auszuwählen, das dann auf der Werkzeugleiste erscheint. Die drei folgenden Funktionen werden auf die spezifizierte Menge von Nachfragestandorten (*specified location set*) angewendet (vgl. Kap. 4.1 Übung 2 Punkt 2 und Punkt 3).

Spezifikation der Menge der vorgegebene Angebotsstandorte (*specify set of fixed supply locations*)

Spezifikation der Menge der ausgeschlossene Angebotsstandorte (*specify set of excluded supply locations*)

 Spezifikation der Menge der Nachfragestandorte der Startkonfikuration (*specify set of starting supply locations*)

Funktionen

Sie werden jeweils auf die Menge der durch das Symbol ( ,  , ) spezifizierten Nachfragestandorte angewendet.

 Anzeige der Nachfragestandorte der aktuellen Menge (s*how specification*),

 Markierte Nachfragestandorte werden zu Standorten der aktuellen Menge (*enter selected locations as set of specified),*

 Löschen aller Standorte der aktuellen Menge (c*lear set of specified locations*).

Anhang B: Programmcode von LOC-ALLOC (C++ Programm)

Zum besseren Verständnis dieses Source Codes werden die wichtigsten hier verwendeten Variablennamen in Beziehung zu den im zweiten Kapitel gegebenen methodischen Grundlagen näher erklärt.

class **ORT**: **Knoten** des Graphen

class **VERBINDUNG**: **Kanten** des Graphen

class **MODELL**: Diese Klasse enthält Variablen in denen die Parameter der Randbedingungen und der Zielfunktion gespeichert sind

class **REGION**: Variablen in denen die Angebotsstandorte und die ihnen zugeordneten Nachfragestandorte gespeichert werden

gewicht: Nachfrage

Programmname: `loc_alloc.cpp`

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <iostream.h>
#include <float.h>
#include <math.h>
#include <errno.h>
#include <time.h>
#include <direct.h>
#include <conio.h>


// Es gibt 4 Klassen in diesem Programm

//
//   1,2. Die 2 Klassen ORT und VERBINDUNG beinhalten die Variablen zur
// Repräsentation des Graphen und Funktionen, die die Eingabe und
// Ausgabe der Graphendaten behandeln
// In der Klasse VERBINDUNG befinden sich auch die Funktionen, die zur
// Berechnung aller kürzesten Wege innerhalb eines Graphen dienen

// 3. In der Klasse MODELL sind die Variablen zur Modellspezifikation gespeichert

// 4.   Für die Berechnung des Standortproblems (Angebotsstandorte und
Zuordnungen) wird // die Klasse REGION geschaffen
// Hier befinden sich alle Variablen und Funktionen die zur Berechnung der
// Regionen benötigt werden

class VERBINDUNG;
class REGION;
class ORT;

class MODELL
   {
```

```
   friend REGION;

   public:

   float kapober,kapunter,maxdis,a1,c1,aexp,bexp,g;
  char nodes[200],vertices[200],nodes_neighbor[200];
   int versuche_sol,versuche,zielfunktion;
   int reg_anz;
   int *vor_geg,*anfang;
   int working();
   int laden(char* ortsfile,char* distanzfile,char* modellfile,char*
   ausgabefile,char* regfile);
   void zeigen();
   void error(int error_nr,char *ausgabe);


   } modell;
int timer=0;
class ORT
   {
   //public:
   friend REGION;
   friend MODELL;
   float gewicht;
   float kapober;
   float kapunter;
   int vorge,ausge,anfa,zuge;
   public:

   static int anzahl;
   static float ges_gewicht;

   int laden(char* filename);
   void zeigen();
   void error(int error_nr,char* ausgabe);


   } *ort;

int ORT::anzahl=0;
float ORT::ges_gewicht=0;

class VERBINDUNG
   {
   friend  class REGION;
    int  *exist;
   float  *distanz;
   float  *interaktion;   // für künftige Erweiterung

   public:
// Funktionen zur Berechnung aller kürzesten Wege innerhalb eines Graphen
       int  min_finden_rek(int *quelle,int v,int ziel,float *mindis,float
       *distanz,int *tiefe,int *ort_reihe);
   float  min_finden(int von,int nach,int *ort_reihe);
   int  komplett();

// laden und speichern der kürzesten Wege
   int  laden(char* filename);
   int  kom_laden(char* filename);
   int  kom_speichern(char* filename);
   void  zeigen();
   void  error(int error_nr, char *ausgabe);
   }  *verbindung ;


class REGION
   {

   int zentrum;
```

```
   float gewicht,gewicht_ab_c1;
   int *orte;
   float *zent_dist;
   public:
   static int anzahl;
   static int vorgeg_zent_anz;

// Initialisierung der Regionsvariablen(zentren,orte)

   int initialisieren(int anz);
   int zufall(int anz);
   int zufallszentren(int vorgeg);
   void orte_loeschen();


// Allgemeine Bearbeitung von Regionen

   int zentrum_eintragen(int zen_nr);
   int welche(int ort,float *zf);
   int ort_eintragen(int ort_nr,float dist);
   int ort_austragen(int ort_nr);
   int zentrum_ort_tauschen(int ort_nr);

   float test_ort_verbun(int ort_nr,int zf);
   float test_ort_verbun_ohne(int ort_nr,int ohne,int zf);


// Zielfunktionsberechnungen auf verschiedenen Ebenen

   float max_find(float w1,float w2);
   float zielfunc(int von,int nach,int zf); //Distanzbedingung B4,
                                            // Zielfunktionen Z1,Z2,Z3
   float zielfunc_reg(int zf);
   float zielfunc_glo(int zf);
   int test_ziel_zent();


// Ausgabe und Speicherung der gefundenen Regionen

   void zeigen();
   void in_file_schreiben(char *ausgabe,char *region,int counter,int *b_a,int
errror);
   void error(int error_nr,char *ausgabe);


// Implementierung des "ALBA" Algorithmuses
   int ort_fin_min(float *dist_zent);
   int alle_ort_max(int *gef_reg,float *dist_zent);
   int alle_ort_min(int *gef_reg,float *dist_zent,int beding_zf);
   int alle_ort_reihen(int *gef_reg,float *dist_zent);
   int bis_kapu_fuellen();
   int alba2(int versuch);
   int alba2X();
   int alba3(int zf);

   int alle_reg_zent();

   } *region;

int REGION::anzahl=0;
int REGION::vorgeg_zent_anz=0;

int REGION::zufall(int anz)
   {
   char str[10],*endptr;
   int i,zz,zzz,r;
   float div,help,ep;
   printf("in Wurzel:",anz);
   gets(str);
```

```
   help=strtod(str,&endptr);
   gets(str);
   ep=atoi(str);//strtod(str,&endptr);//1/(atoi(str));
        ep=1/ep;
         div=pow(help,ep);
   printf(" %f hoch %f = %f\n",help,ep,div);
   gets(str);
       return -1;
      }


float REGION::max_find(float w1,float w2)
   {
   if(w1>=w2)
      return w1;
   else
      return w2;
   }
int REGION::zufallszentren(int vorgeg)
   {
   int o,r,zz,r2,zzz;
   char str[6];
   for(r=1;r<=vorgeg;r++)
      ort[region[r].zentrum].anfa=1;
   for(o=1;o<=ort->anzahl;o++)
      if((ort[o].anfa!=1)&&(ort[o].vorge!=1))
         ort[o].zuge=0;
   for(r=1;r<=region->anzahl;r++)
      {
      for(o=1;o<=ort->anzahl;o++)

         {
         region[r].orte[o]=-1;
         region[r].zent_dist[o]=0;
         }
      if((ort[region[r].zentrum].anfa!=1)&&(ort[region[r].zentrum].vorge!=1))
         region[r].gewicht=0;
      else
         region[r].gewicht=ort[region[r].zentrum].gewicht;
      }
   for(r=vorgeg+1;r<=region->anzahl;r++)
      {
      zz=-1;
      zzz=1;//Wenn unter 10 000 Zufallszahlen kein passendes Zentrum gefunden wird
Abbruch
      while((zz==-1)&&(zzz<10000))
         {
         zz=(rand()%ort->anzahl)+1;
         for(r2=1;r2<r;r2++)
            if((region[r2].zentrum==zz)||(ort[zz].ausge==1))
               zz=-1;
         zzz++;
         }
      if (zz!=-1)
         region[r].zentrum_eintragen(zz);
      else
         return -1;
      }
   for(o=1;o<=ort->anzahl;o++)
         ort[o].anfa=0;
   for(r=1;r<=region->anzahl;r++)
      modell.anfang[r]=region[r].zentrum;
   return 1;
   }
int REGION::initialisieren(int anz)
   {
   int r,o,i=1,anfangs_zent=0;
   char str[3];
   srand((unsigned) time (NULL));
```

```
   region->anzahl=anz;
   region = new REGION[anz+1];
   for(r=1;r<=anz;r++)
      {
      while((ort[i].vorge==0)&&(ort[i].anfa==0)&&(i<ort->anzahl))
         i++;
      if((ort[i].vorge==0)&&(ort[i].anfa==0))
           {
         region[r].zentrum=-1;
         region[r].gewicht=0;
         }
      else
         {
         region[r].gewicht=0;

         if(ort[i].anfa==0)
            region->vorgeg_zent_anz++;
         else
            anfangs_zent++;
         i++;
         }
      if(!(region[r].orte= new int[ORT::anzahl+2]))
         return -9;//region->error(-9,"c:loc_error");
      if(!(region[r].zent_dist= new float[ORT::anzahl+2]))
         return -10;//region->error(-10,"c:loc_error");;

      for(o=1;o<=ORT::anzahl+1;o++)
      {
         region[r].orte[o]=-1;
         region[r].zent_dist[o]=0;
         }
      }
   for(r=1;r<=region->vorgeg_zent_anz;r++)
         {
         region[r].zentrum_eintragen(modell.vor_geg[r]);
         }
   for(r=1;r<=anfangs_zent;r++)
         {
         region[region->vorgeg_zent_anz+r].zentrum_eintragen(modell.anfang[r]);
         }
   REGION::anzahl=anz;
   region->vorgeg_zent_anz+=anfangs_zent;
   if(region->zufallszentren(region->vorgeg_zent_anz)==-1)
      return -1;
   return 1;
   }
void REGION::zeigen()
   {
   int r,o;
   FILE *fi;
   printf("%d Regionsanzahl \n",region->anzahl);
   for(r=1;r<=region->anzahl;r++)
      {
      printf(" Region %d  Gesamtgewicht %f Gesamtdistanz %f mit Zentrum %d
      bestehend aus Orten :
      ",r,region[r].gewicht,region[r].zielfunc_reg(modell.zielfunktion),region[r].
      zentrum);
      o=1;
      while(region[r].orte[o]!=-1)
         printf("%d  ",region[r].orte[o++]);
      printf("\n");
      }
   }
void REGION::orte_loeschen()
   {
   int r,o;
   for(r=1;r<=region->anzahl;r++)
      {
      o=1;
```

```
     while(region[r].orte[o]!=-1)
        {
        region[r].orte[o]=-1;
        region[r].zent_dist[o]=0;
       o++;
        }
     region[r].gewicht=0;
     }
   for(o=1;o<=ort->anzahl;o++)
     ort[o].zuge=0;
   for(r=1;r<=region->anzahl;r++)
     region[r].zentrum_eintragen(modell.anfang[r]);
   }
void REGION::in_file_schreiben(char *modell,char *ausgreg,int counter,int
*b_a,int error)
   {
   char str[4];
   int r,o,*o_anz,ot,zt;
   float ges_gew=0,zw_zf1=0,zw_zf2=0,zw_zf3=0,max_zf1,ges_zf2=0,ges_zf3=0;
   FILE *fi;
   FILE *ausgabe_file;
   o_anz= new int[region->anzahl+1];
     max_zf1=0;
     for(r=1;r<=region->anzahl;r++)
        {
        zw_zf1=region[r].zielfunc_reg(1);
        zw_zf2=region[r].zielfunc_reg(2);
        zw_zf3=region[r].zielfunc_reg(3);
        if(zw_zf1>max_zf1)
           max_zf1=zw_zf1;
        ges_zf2+=zw_zf2;
        ges_zf3+=zw_zf3;
        ges_gew+=region[r].gewicht;
        }
    if ((ausgabe_file=fopen(modell,"a"))!=NULL)
        {
        fprintf(ausgabe_file,"\n********** solution **********\n");
        fprintf(ausgabe_file,"with starting locations : ");
        for(r=1;r<=region->anzahl;r++)
           fprintf(ausgabe_file,"%d,",b_a[r]);
        fprintf(ausgabe_file,"\n");
        fprintf(ausgabe_file,"solution= maximum distance=%f,demand over c1
        distance=%f,average distance/demand unit=%f\n",
        max_zf1,(ges_zf2*ORT::ges_gewicht)/region-
>anzahl,ges_zf3/ORT::ges_gewicht);
        fclose(ausgabe_file);
        }
   fi=fopen(ausgreg,"w+");
   fprintf(fi,"
   s_area,supply_nr,node_nr,is_supply,node_count,demand,obj_1,obj_2,obj_3\n");
   fprintf(fi,"0,0,0,1,%d,%f,%f,%f,%f\n",
   ORT::anzahl,ges_gew/region->anzahl,max_zf1,(ges_zf2*ORT::ges_gewicht)/region-
>anzahl,
   ges_zf3/ORT::ges_gewicht);
   for(r=1;r<=region->anzahl;r++)
     {
     o=1;
     o_anz[r]=0;
     zt=region[r].zentrum;
     while(region[r].orte[o]!=-1)
        {
        o_anz[r]++;
        ot=region[r].orte[o];
        fprintf(fi,"%d,%d,%d,0,0,%f,%f,%f,%f\n"
        ,r,region[r].zentrum,ot,ort[ot].gewicht,region->zielfunc(ot,zt,1),
        region->zielfunc(ot,zt,2)*ORT::ges_gewicht,
        region->zielfunc(ot,zt,3)/ort[ot].gewicht);
        o++;
        }
```

```
      fprintf(fi,"%d,%d,%d,1,%d,%f,%f,%f,%f\n"
      ,r,region[r].zentrum,region[r].zentrum,o_anz[r],region[r].gewicht,
      region[r].zielfunc_reg(1),region[r].zielfunc_reg(2)*ORT::ges_gewicht,
      region[r].zielfunc_reg(3)/region[r].gewicht);

      }
   fclose(fi);
   }
int REGION::zentrum_eintragen(int zen_nr)
   {
   this->zentrum=zen_nr;//region[reg_nr].zentrum=zen_nr;
   this-
>gewicht+=ort[zen_nr].gewicht;//region[reg_nr].gewicht+=ort[zen_nr].gewicht;
   ort[zen_nr].zuge=1;
   return 1;
   }

int REGION::ort_eintragen(int ort_nr,float dist)
   {
   int o=1;
   while(this->orte[o]!=-1)
      o++;
   this->orte[o]=ort_nr;
   this->zent_dist[o]=dist;
   this->gewicht+=ort[ort_nr].gewicht;
   ort[ort_nr].zuge=1;
   return 1;
   }

float REGION::zielfunc(int von,int nach,int zf) //Distanzbedingung B4,B5 und
Zielfunktionen Z1,Z2,Z3
   {
   float h1;
  char str[3];
   if((von<1)||(von>ORT::anzahl)||(nach<1)||(nach>ORT::anzahl))
      {

      printf("falsche orte für zielfunktion von=%d zent=%d ort:anz
      =%d",von,nach,ORT::anzahl);
      gets(str);
      return -1;
      }
  timer++;
   if(timer==2000)
      {
      timer=0;
      modell.working();
    }
   if(modell.maxdis>verbindung[von].distanz[nach]) // B4 Maximale Distanz
      {
      if(modell.a1>0)      // B5 Maximaler Anteil von a1 Benutzern über Distanz c1
         if((verbindung[von].distanz[nach] > modell.c1)&&
            ((this->gewicht_ab_c1+ort[von].gewicht) > modell.a1))
            return FLT_MAX;
      if(zf==0)   // Z0
            return verbindung[von].distanz[nach];
      if(zf==2)   // Z2
         {
         if(verbindung[von].distanz[nach] >= modell.c1)
            return ort[von].gewicht/ORT::ges_gewicht;
         else
            return 0;
          }
      if(zf==1)
         return verbindung[von].distanz[nach];  // Z1 Minimiere maximale Distanz
      if(zf==3)
         {
         if(verbindung[von].distanz[nach] > modell.g) // Z3
            {
```

```
   h1=pow(ort[von].gewicht,modell.aexp)*pow(verbindung[von].distanz[nach],modell.
bexp);
           return h1;
           }
        else
           return 0;
         }
     if(zf==4)
        {
        if(verbindung[von].distanz[nach] > modell.c1)
           return 0;
        return -pow(ort[von].gewicht,modell.aexp);
        }
     return FLT_MAX;
     }
   else
      return FLT_MAX;
   }

float REGION::zielfunc_reg(int zf)
   {
   int o=1;
   float max_dist=0,dist=0,summe=0;
   //printf("start zielfunc_reg reg zent=%d \n",this->zentrum);
   while(this->orte[o]!=-1)
      {
      //printf("vor zielfunc aufruf o=%d ort=%d zent=%d \n",o,this->orte[o],this-
>zentrum);
      dist=this->zielfunc(this->orte[o],this->zentrum,zf);
      if((dist==FLT_MAX)||(summe==FLT_MAX))
          summe=FLT_MAX;
     else
          summe+=dist;
      if(dist>max_dist)
         max_dist=dist;
      //ges_dist+=this->zent_dist[o];
      o++;
      }
   //printf("stop zielfunc_reg\n");
   if(zf==1)
      return max_dist;
   else
      return summe;
   }

float REGION::zielfunc_glo(int zf)
   {
   int r=1;
   float max_dist=0,dist=0,summe=0;
   for(r=1;r<=region->anzahl;r++)
      {
      dist=region[r].zielfunc_reg(zf);
      if((dist==FLT_MAX)||(summe==FLT_MAX))
          summe=FLT_MAX;
     else
          summe+=dist;
      if(dist>max_dist)
         max_dist=dist;
      }
   if(zf==1)
      return max_dist;
   else
      return summe;
   }
int REGION::test_ziel_zent()    //testet ob der wechsel des zentrums innerhalb
einer region
   {                            // die zielfunktion verbessert
    int o=1,change=0;
```

```
   float min_zielwert,help;
   if (ort[this->zentrum].vorge==1)
      return 0;
   min_zielwert=this->zielfunc_reg(modell.zielfunktion); //Zielfunktion der
aktuellen Region
   while(this->orte[o]!=-1)     //alle Orte der Region werden getestet
      {
      if(ort[this->orte[o]].ausge==0)
         if((ort[this->orte[o]].kapunter<=this->gewicht)&&
            (ort[this->orte[o]].kapober>=this->gewicht))
         {
         this->zentrum_ort_tauschen(o);

         help=this->zielfunc_reg(modell.zielfunktion);
         if(help<min_zielwert)
            {
            min_zielwert=help;
            change=o;
            }
         this->zentrum_ort_tauschen(o);
         }
       o++;
      }
   return change;
   }

int REGION::alba3(int zf)
   {
   int
r=1,r2=1,o=1,o2,tausch_ort=0,tausch_ort2,zen,zen2,gef_reg=0,gef_ort=0,nochmal=1,c
ounter=0;
   float mindis=0,help=0,dist_r2,dist_r1,k1,k2,k3,k4;
   float dist_r2_t2,dist_r1_t1,dist_r1_t2,dist_r2_t1,ku1,ku2,ko1,ko2;//variablen
paarweise tauschen
  char str[12];


   while(nochmal==1)  //&&(counter<100))     //solange bis nichts mehr zu machen
ist       {
   {
      nochmal=0;
      counter++;
      for(r=1;r<=region->anzahl;r++)// für jede Region
      if((gef_ort=region[r].test_ziel_zent())!=0)//testet ob Zentrumwechsel
sinnvoll
         {
         nochmal=1;
         region[r].zentrum_ort_tauschen(gef_ort);
         }
      for(r=1;r<=region->anzahl;r++)// für jede Region
      {
         for(r2=1;r2<=region->anzahl;r2++)         // für jede andere Region
            {
            o=1;  //
            while(region[r].orte[o]!=-1)         // für allle Regionen der 1.Region
               {
               if (r!=r2)
                  {
                  tausch_ort=region[r].orte[o];
                  zen=region[r].zentrum;
                  zen2=region[r2].zentrum;
                  if((dist_r2=region[r2].test_ort_verbun(tausch_ort,zf))==-1)
                     {
                     o++;
                     continue;
                     }
                  if((dist_r1=region->zielfunc(tausch_ort,zen,zf)) < dist_r2)
                     {
                     o++;
```

```
                  continue;
                  }
               if(dist_r1==dist_r2)
                  {
                  o++;
                  continue;
                  }

               if(((k1=region[r].gewicht-
               ort[tausch_ort].gewicht)>=(k2=ort[zen].kapunter))&&

   ((k3=region[r2].gewicht+ort[tausch_ort].gewicht)<=(k4=ort[zen2].kapober)))
                  {//wenn herausnahme von tausch_ort nicht region r unter
kapunter fallen läßt
            if(region[r].ort_austragen(tausch_ort)<0)
                     {

                      }    //if ort austragen=-1
                  else
                       {
                     region[r2].ort_eintragen(tausch_ort,dist_r2);
                     nochmal=1;
                      }
                  } //if kapazitätsrestriktionen
               }//if (r!=r2)
             o++;
             }//while(region[r].orte[o]!=-1)
          } // for r2
       } // for r1
     if(nochmal==0)
        {     //anfang paarweises austauschen
        for(r=1;r<=region->anzahl;r++)// für jede Region
           {
           o=1;
           zen=region[r].zentrum;
           while(region[r].orte[o]!=-1)   // für alle Orte der 1.Region
              {
              tausch_ort=region[r].orte[o];
              for(r2=1;r2<=region->anzahl;r2++) // für jede andere Region
           if(r!=r2)
                {
                o2=1;
                while((region[r2].orte[o2]!=-1)&&(nochmal==0)) // für alle Orte
   der
                                                                //2.Region
             {
                  tausch_ort2=region[r2].orte[o2];
                  zen2=region[r2].zentrum;
   if((dist_r2_t1=region[r2].test_ort_verbun_ohne(tausch_ort,tausch_ort2,zf))==-
1)
                     {
                     o2++;
                     continue;
                     }

   if((dist_r1_t2=region[r].test_ort_verbun_ohne(tausch_ort2,tausch_ort,zf))==-1)
                     {
                     o2++;
                     continue;
                     }
                  k1=region[r].gewicht-
ort[tausch_ort].gewicht+ort[tausch_ort2].gewicht;
                  ku1=ort[zen].kapunter;
                  ko1=ort[zen].kapober;
                  k2=region[r2].gewicht+ort[tausch_ort].gewicht-
ort[tausch_ort2].gewicht;
                  ku2=ort[zen2].kapunter;
                  ko2=ort[zen2].kapober;
                  if((k1>=ku1)&&(k1<=ko1)&&(k2>=ku2)&&(k2<=ko2))
```

```
                {
                     dist_r1_t1=region-
>zielfunc(tausch_ort,zen,zf);//modell.zielfunktion
                     dist_r2_t2=region->zielfunc(tausch_ort2,zen2,zf);
                     if(modell.zielfunktion==1)

   if(max_find(dist_r1_t1,dist_r2_t2)<=max_find(dist_r1_t2,dist_r2_t1))
                          {
                          o2++;
                          continue;
                          }
                     if(modell.zielfunktion!=1)
                   {
                        if((dist_r1_t1==FLT_MAX)||(dist_r2_t2==FLT_MAX))
                          dist_r1=FLT_MAX;
                        else
                          dist_r1=dist_r1_t1+dist_r2_t2;
                        if((dist_r1_t2==FLT_MAX)||(dist_r2_t1==FLT_MAX))
                          dist_r2=FLT_MAX;
                        else
                          dist_r2=dist_r1_t2+dist_r2_t1;
                        if((dist_r1)<=(dist_r2))
                          {
                          o2++;
                          continue;
                          }
                   }
                     region[r].ort_austragen(tausch_ort);
                     region[r2].ort_austragen(tausch_ort2);
                     region[r].ort_eintragen(tausch_ort2,dist_r1_t2);
                     region[r2].ort_eintragen(tausch_ort,dist_r2_t1);

                     nochmal=1;
                     } //if kapazitätsrestriktionen
                    else
                {
                     o2++;
                     continue;
                     }
                    }//while(region[r2].orte[o2]!=-1)
                }//if (r!=r2)
              o++;
              }//while(region[r].orte[o]!=-1)
            } // for r1
         }// ende paarweises austauschen
      } // while nochmal
   for(r=1;r<=region->anzahl;r++)  // für jede Region
      {
      if(region[r].zielfunc_reg(1)==FLT_MAX)
         {
         return -4;
       }
      }
   return 1;
   }
    // func alba3

int REGION::zentrum_ort_tauschen(int t_ort)//macht den an der stelle o
befindlichen Ort
   {                                        // zum zentrum der region
   REGION *help;
   int zen_alt,o=1;
   if(this->orte[t_ort]==-1)
      return -1;
   zen_alt=this->zentrum;
   this->zentrum=this->orte[t_ort];
   this->orte[t_ort]=zen_alt;
   while(this->orte[o]!=-1)
      {
```

```
     this->zent_dist[o]=region->zielfunc(this->orte[o],this-
>zentrum,modell.zielfunktion);
     o++;
     }

   return 1;
   }



int REGION::ort_austragen(int ort_nr)//trägt ort aus region aus, falls kapunter
nicht unterschritten wird
   {
   int o=1,ort_in_reg;
   while((this->orte[o]!=ort_nr)&&(this->orte[o]!=-1))
      o++;
   if(this->orte[o]==-1)
      return -2;
   ort_in_reg=o;
    o=1;
   while(this->orte[o]!=-1)
      o++;
   this->orte[ort_in_reg]=this->orte[o-1];
   this->orte[o-1]=-1;
   this->gewicht-=ort[ort_nr].gewicht;
   ort[ort_nr].zuge=0;
   return 1;
   }


float REGION::test_ort_verbun(int ort_nr,int zf)//testet ob ort_nr mit
aufgerufener region verbunden
   {                                      // ist und liefert distanz zum zentrum
zurück
   int i=1,o=1,gef_ort=-1;
   float mindis=0;
   if(verbindung[this->zentrum].exist[ort_nr])
      return region->zielfunc(ort_nr,this->zentrum,zf);
      //mindis=verbindung[this->orte[i]].distanz[ort_nr];
   while(this->orte[i]!=-1)
      {
      if(verbindung[this->orte[i]].exist[ort_nr])
      {
        return region->zielfunc(ort_nr,this->zentrum,zf);
       }
      i++;
      }
   return -1;
   }
float REGION::test_ort_verbun_ohne(int ort_nr,int ohne,int zf)//testet ob ort_nr
mit
             // aufgerufener region ohne ORT "ohne" verbunden
   {                                      // ist und liefert distanz zum zentrum
zurück
   int i=1,o=1,gef_ort=-1;
   float mindis=0;
   if(verbindung[this->zentrum].exist[ort_nr])
      return region->zielfunc(ort_nr,this->zentrum,zf);//modell.zielfunktion
      //mindis=verbindung[this->orte[i]].distanz[ort_nr];
   while(this->orte[i]!=-1)
      {
      if((verbindung[this->orte[i]].exist[ort_nr])&&(this->orte[i]!=ohne))
         {
          return region->zielfunc(ort_nr,this->zentrum,zf);
           }
      i++;
       }
   return -1;
   }
```

```
int REGION::ort_fin_min(float *dist_zent)//Findet Ort, dessen Zielfunktionswert
zur
                                        // aufgerufenen Region
   {                    // minimal ist, der verbunden ist, und die
Kapazitätsobergrenze
   int i=1,o=1,ort_nr=-1,beding_zf=0;        // der Region nicht überschreittet
   float help,mindis=0,gesgew;
   for(i=1;i<=ort->anzahl;i++)
      {
      if(!(ort[i].zuge))        // nicht zugewiesener Ort
         {
         if((ort[i].gewicht+this->gewicht)<=(ort[this->zentrum].kapober))//wenn
                                    //Kapobergrenze nicht
            {               // überschritten wird
            help=this->test_ort_verbun(i,beding_zf);
            if((help!=-1)&&((mindis==0)||(help<mindis)))
               {
               mindis=help;
               ort_nr=i;
                 }
             }
         }
      }
   *dist_zent=mindis;
   return ort_nr;
   }
int REGION::bis_kapu_fuellen()
   {
   char str[4];
   int i,gef_ort,end=0,dummy,r;
   float dist_zent=0;
   while (!end)
      {
end=1;
      for(i=1;i<=region->anzahl;i++)
         if((region[i].gewicht)<(ort[region[i].zentrum].kapunter))
            {
            gef_ort=region[i].ort_fin_min(&dist_zent);
            if(gef_ort==-1)
               return -1;
            region[i].ort_eintragen(gef_ort,dist_zent);
            end=0;
            }
      }
   return 1;
   }
int REGION::alle_ort_min(int *gef_reg,float *dist_zent,int beding_zf) //Sucht
Ort, der minimale Distanz zum nächsten Zentrum hat und der
   { // die Kapazitätsobergrenze der Region nicht überschreitet
   char str[3];
   int i=1,r=1,ort_nr=-1;
   float help,mindis=0,gesgew;
   for(i=1;i<=ORT::anzahl;i++)
      {
      if(!(ort[i].zuge))                //alle nicht zugeordneten Orte
         {
         for(r=1;r<=region->anzahl;r++)       //alle Regionen
            {

   if((ort[i].gewicht+region[r].gewicht)<=(ort[region[r].zentrum].kapober))
               {
               help=region[r].test_ort_verbun(i,beding_zf);
               if((help!=-1)&&((mindis==0)||(help<mindis)))
                  {
                  mindis=help;
                  ort_nr=i;
                   *gef_reg=r;
                  }
                  }
```

```
          }
        }
      }

   *dist_zent=mindis;
   return ort_nr;
   }
int REGION::alle_ort_reihen(int *gef_reg,float *dist_zent) //Sucht Ort, der
minimale Distanz zum nächsten Zentrum hat und der
   { // die Kapazitätsobergrenze der Region nicht überschreitet
   char str[3];
   int i=1,r=1,ort_nr=-1,beding_zf=0;
   float help,mindis=0,gesgew;
   for(i=1;i<=ORT::anzahl;i++)
      {
      if(!(ort[i].zuge))               //alle nicht zugeordneten Orte
         {
         mindis=-1;
       *gef_reg=0;
         for(r=1;r<=region->anzahl;r++)       //alle Regionen
            {
   if((ort[i].gewicht+region[r].gewicht)<=(ort[region[r].zentrum].kapober))
               {
               help=region[r].test_ort_verbun(i,beding_zf);
               if((help!=-1)&&((mindis==-1)||(help<mindis)))
                  {
                  mindis=help;
                  ort_nr=i;
                   *gef_reg=r;
                  }
            }
            }
         if(mindis!=-1)  //wurde eine region für Ort i gefunden->eintragen
            {
          *dist_zent=mindis;
            return i;
         }
         }
      }
   return -1;
   }
int REGION::welche(int ort,float *zf)
   {
   int found=0,r,o;
   for(r=1;r<=REGION::anzahl;r++)
      for(o=1;o<=ORT::anzahl;o++)
         if(region[r].orte[o]==ort)
            {
            *zf=region[r].zent_dist[o];
            return r;
            }
   return -1;
   }
int REGION::alba2X()
   {
   char str[5];
   int *unversorgt;
   int *versorgt;
   int o,r,uv_anz=0,v_anz=0,t=1,uv,v,reg,hilf,tausch_max,gef_ort,beding_zf=0;
   float uv_ort_zf,v_ort_zf,zent_zf;
   tausch_max=ORT::anzahl*10;//pow(ORT::anzahl,2)*10;
   unversorgt = new int[ORT::anzahl+1];
   versorgt = new int[ORT::anzahl+1];
   for(o=1;o<=ORT::anzahl;o++)
      if(ort[o].zuge==1)
         {
         v_anz++;
         versorgt[v_anz]=o;
         }
```

```
      else
         {
         uv_anz++;
         unversorgt[uv_anz]=o;
         }
   while((uv_anz>0)&&(t<tausch_max))
      {
      uv=(rand()%uv_anz)+1;
      v=(rand()%v_anz)+1;
      if((reg=region->welche(versorgt[v],&v_ort_zf))==-1)
           return -3;
      uv_anz=%d\n",reg,uv,unversorgt[uv],uv_anz);
      if((uv_ort_zf=region[reg].test_ort_verbun(unversorgt[uv],beding_zf))!=-1)
         if(ort[unversorgt[uv]].gewicht < ort[versorgt[v]].gewicht)//&& (v_ort_zf
> (uv_ort_zf)))
           {
           region[reg].ort_austragen(versorgt[v]);
           region[reg].ort_eintragen(unversorgt[uv],uv_ort_zf);
           hilf=versorgt[v];
           versorgt[v]=unversorgt[uv];
           unversorgt[uv]=hilf;
           while((gef_ort=region->alle_ort_min(&reg,&zent_zf,0))!=-1)
              {
              region[reg].ort_eintragen(gef_ort,zent_zf);
              uv=1;
              while(gef_ort!=unversorgt[uv])
                 uv++;
              unversorgt[uv]=unversorgt[uv_anz];
              uv_anz--;
              v_anz++;
              versorgt[v_anz]=gef_ort;
              }
           }
      uv=0;
      v=0;
      for(o=1;o<=ort->anzahl;o++)
      if(ort[o].zuge==1)
         {
         v++;

         }
      else
         {
         uv++;
         }
      t++;
      }
   if(uv_anz>0)
      return-1;
   return 1;
   }

int REGION::alba2(int versuch)    //
   {
   char str[4];
   int error=-1,gef_ort,gef_reg,i,v=1,r,max_versuche;
   float dist_zent;
   while((error==-1)&&(v<=modell.versuche_sol))
      {
      if((error=region->bis_kapu_fuellen())==-1)
         {
         if(region->vorgeg_zent_anz==region->anzahl)
            v=modell.versuche_sol;
         v++;
         region->orte_loeschen();
         region->zufallszentren(region->vorgeg_zent_anz);
         }
      }
   if(error==-1)
```

```
      {
      return -2;
      }
   switch(versuch)
      {
      case 1:
         while((gef_ort=region->alle_ort_min(&gef_reg,&dist_zent,0))!=-1)
            region[gef_reg].ort_eintragen(gef_ort,dist_zent);
         break;
      case 2:
         while((gef_ort=region->alle_ort_reihen(&gef_reg,&dist_zent))!=-1)
            region[gef_reg].ort_eintragen(gef_ort,dist_zent);
         break;
      case 3:
         while((gef_ort=region-
>alle_ort_min(&gef_reg,&dist_zent,modell.zielfunktion))!=-1)
            region[gef_reg].ort_eintragen(gef_ort,dist_zent);
         break;
    }

    error=0;

   for(i=1;i<=ort->anzahl;i++)
      if(ort[i].zuge!=1)
         {
         error=-2;
         }
   if(error==-2)
      if((error=alba2X())<0)
         {
         return -3;
         }
   return 1;
   }
void REGION::error(int error_nr,char *ausgabe)
   {
   char error_str[100];
   FILE *ausgabe_file;
   switch(error_nr)
      {
      case -1:
         strcpy(error_str,"couldn't find centers for regions");
         break;
      case -2:
         strcpy(error_str,"couldn't fill regions up to minimum capacity");
         break;
      case -3:
         strcpy(error_str,"couldn't fit regions under maximum capacity");
         break;
      case -4:
         strcpy(error_str,"couldn't fulfill distance constraints");
         break;
      case -5:
         strcpy(error_str,"no trials made to solve problem");
         break;
      case -9:
         strcpy(error_str,"Out of memory 1");
         break;
      case -10:
         strcpy(error_str,"Out of memory 2");
         break;
      case -11:
         strcpy(error_str,"Out of memory 3");
         break;
      }
      if ((ausgabe_file=fopen(ausgabe,"a"))!=NULL)
         {
         fprintf(ausgabe_file,"\n********** ERROR **********\n",error_str);
         fprintf(ausgabe_file,"error=regions : %s \n",error_str);
```

```
        fclose(ausgabe_file);
        }
  exit(-1);
   }
void MODELL::error(int error_nr,char *ausgabe)
   {
   char error_str[100];
   FILE *ausgabe_file;
   switch(error_nr)
      {
      case -1:
         strcpy(error_str,"no file with model formulations");
         break;
      case -2:
         strcpy(error_str,"no number_of_regions parameter");
         break;
      case -3:
         strcpy(error_str,"no objective parameter");
         break;
      case -9:
         strcpy(error_str,"Out of memory1");
         break;
      case -10:
         strcpy(error_str,"Out of memory2");
         break;
      case -11:
         strcpy(error_str,"Out of memory3");
         break;
      }
      if ((ausgabe_file=fopen(ausgabe,"a+"))!=NULL)
         {
         fprintf(ausgabe_file,"\n********** ERROR **********\n",error_str);
         fprintf(ausgabe_file,"error=model :  %s \n",error_str);
         fclose(ausgabe_file);
         }
  exit(-1);
   }
int VERBINDUNG::kom_laden(char* filename)
   {
   char verbindung_str[100],*endptr;
   int i=1,j=1,von,nach;
   FILE *file;
    double help;
   if ((file=fopen(filename,"r"))!=NULL)
      {
      fgets(verbindung_str,100,file);
      while (!feof(file))
         {
         if((fgets(verbindung_str,100,file))!=NULL)
            {
            timer++;
            if(timer==2000)
               {
               timer=0;
               modell.working();
            }
            von=atoi( strtok(verbindung_str,","));
            nach=atoi( strtok(0,","));
            if((von>ORT::anzahl)||(von<1)||(nach>ORT::anzahl)||(nach<1))
               return -6;
            help=strtod(strtok(0,","),&endptr);
            verbindung[von].distanz[nach]=help;
            verbindung[nach].distanz[von]=help;//verbindung[von].distanz[nach];
            }
         }
      fclose(file);
     timer=0;
      return 1;
      }
```

```
   else
      {
      return -1;
       }
   }
int VERBINDUNG::laden(char* filename)
   {
   char verbindung_str[100],*endptr;
   int i=1,j=1,von,nach;
   FILE *file;
    double help;
   if ((file=fopen(filename,"r"))!=NULL)
      {
      if(!(verbindung = new VERBINDUNG[ORT::anzahl+1]))
          return -9;//verbindung->error(-9,"c:loc_error");
      for(i=1;i<=ort->anzahl;i++)
         {
         if(!(verbindung[i].exist= new  int[ORT::anzahl+1]))
            return -10;//verbindung->error(-10,"c:loc_error");
         if(!(verbindung[i].distanz= new  float[ORT::anzahl+1]))
            return -11;//verbindung->error(-11,"c:loc_error");
         for(j=1;j<=ORT::anzahl;j++)
            {
            verbindung[i].exist[j]=0;
            verbindung[i].distanz[j]=-1;
            if(i==j)
               verbindung[i].distanz[j]=0;
            }
          }
      fgets(verbindung_str,100,file);
      while (!feof(file))
         {
         if((fgets(verbindung_str,100,file))!=NULL)
            {
            von=atoi( strtok(verbindung_str,","));
            nach=atoi( strtok(0,","));
            if((von>ORT::anzahl)||(von<1)||(nach>ORT::anzahl)||(nach<1))
               return -2;
            verbindung[von].exist[nach]=1;
            verbindung[nach].exist[von]=1;
            help=strtod(strtok(0,","),&endptr);
            verbindung[von].distanz[nach]=help;
            verbindung[nach].distanz[von]=verbindung[von].distanz[nach];
            }
         }
      fclose(file);
      for(i=1;i<=ort->anzahl;i++)
         for(j=1;j<=ORT::anzahl;j++)
            if(verbindung[i].distanz[j]==-1)
               return 0;    // Verbindungsmatrix nicht komplett
      return 1; // Verbindungsmatrix  komplett
      }
   else
      {
      return -1;
      }
   }
int VERBINDUNG::kom_speichern(char* filename)
   {
   int i,j;
   char string[5];
   FILE *kom_file;
   kom_file=fopen(filename,"w+");
   fprintf(kom_file,"von,nach,Distanz\n");
   for(i=1;i<=ORT::anzahl;i++)//ORT::anzahl;i++)
      for(j=1;j<=ORT::anzahl;j++)
         if(i<j)
            fprintf(kom_file,"%d,%d,%f\n",i,j,verbindung[i].distanz[j]);
   fclose(kom_file);
```

```
  return 1;
   }
void VERBINDUNG::zeigen()
   {
   int i,j;
   char string[5];
   for(i=1;i<=3;i++)//ORT::anzahl;i++)
      {
      for(j=1;j<=ORT::anzahl;j++)
         if(i<j)
         {
         //if(verbindung[i].exist[j]!=0)
            printf("von %d nach %d : Distanz %f Interaktion \n"
            ,i,j,verbindung[i].distanz[j]);//,verbindung[i].interaktion[j]) ;
            if(j==20)
               gets(string);
         }
       gets(string);
        }
   }
int VERBINDUNG::min_finden_rek(int *quelle,int v,int ziel,float *mindis,
                               float *distanz,int *tiefe,int *ort_reihe)
   {
   int i,j,stop;        //findet minimale Verbindung
   *tiefe=*tiefe+1;
   for(i=1;i<=ort->anzahl;i++)
      if((i==ziel)&&(verbindung[v].exist[i]))
         if((*mindis==-1)||(*mindis>*distanz+verbindung[v].distanz[i]))
              {
              *mindis=*distanz+verbindung[v].distanz[i];
              break;
                }
   for(i=1;i<=ort->anzahl;i++)

      if((verbindung[v].exist[i])&&(i!=ort_reihe[*tiefe-1]))
         {
          ort_reihe[*tiefe]=i;
         stop=0;
         for(j=0;j<*tiefe;j++)
            if(ort_reihe[j]==i)
               {
               stop=1;
               break;
                 }
         if((stop==1)||((*mindis!=-
1)&&(*mindis<*distanz+verbindung[v].distanz[i])))
            continue;
         else
            if((verbindung[*quelle].distanz[i]==-
1)||(verbindung[*quelle].distanz[i]>=*distanz+verbindung[v].distanz[i]))
                {
                verbindung[*quelle].distanz[i]=*distanz+verbindung[v].distanz[i];
                   }
               else
                  continue;
            *distanz+=verbindung[v].distanz[i];
            min_finden_rek(quelle,i,ziel,mindis,distanz,tiefe,ort_reihe);
            *distanz-=verbindung[v].distanz[i];
            *tiefe=*tiefe-1;;
          //  }
         }
   return ziel;
   }

float VERBINDUNG::min_finden(int von,int nach,int *ort_reihe)
   {
   int i,j,quelle,tiefe=0 ;          // sucht alle Wege von -> nach
   float mindis=-1,distanz=0,help=0;
```

```
   for(i=1;i<=ort->anzahl;i++)            // für alle Orte
      ort_reihe[i]=0;
   ort_reihe[0]=von;
     quelle=von;
   min_finden_rek(&quelle,von,nach,&mindis,&distanz,&tiefe,ort_reihe);
   verbindung[von].distanz[nach]=mindis;
   return mindis;
   }


int VERBINDUNG::komplett()          // ruft für alle Orte (i nach j) min_finden
auf
   {
   int i,j,error=1;
   float min_dist,help;
   char str[10];
   int *ort_reihe;
   ort_reihe=new int[ort->anzahl+1];
   for(i=1;i<=ort->anzahl;i++)
      {
      for(j=1;j<=ort->anzahl;j++)
         if(j>i)
            {
            //printf("verbindung von %d nach %d suchen \n",i,j);
            verbindung->min_finden(i,j,ort_reihe);
             }
         }
   for(i=1;i<=ort->anzahl;i++)                // füllt über Hauptdiagonale der
Verbindungsmatrix
      for(j=1;j<=ort->anzahl;j++)
         if(i>j)
            {
            verbindung[i].distanz[j]=verbindung[j].distanz[i];
            if (verbindung[i].distanz[j]==-1)
               error=-3;
            if (verbindung[i].distanz[j]==0)
               {
               printf("verbindung von %d nach %d = 0000 \n",i,j);
               gets(str);
               }
            }
   return error;
   }
void VERBINDUNG::error(int error_nr,char *ausgabe)
   {
   char error_str[100],str[4];
   FILE *ausgabe_file;
   switch(error_nr)
      {
      case -1:
         strcpy(error_str,"file with vertices distances not found");
         break;
      case -2:
         strcpy(error_str,"data inside vertices distance file out of range ");
         break;
      case -3:
         strcpy(error_str,"couldn't calculate all missing distances");
         break;
      case -9:
         strcpy(error_str,"Out of memory1");

         break;
      case -10:
         strcpy(error_str,"Out of memory2");

         break;
      case -11:
         strcpy(error_str,"Out of memory3");
```

```
         break;
      }
      if ((ausgabe_file=fopen(ausgabe,"a"))!=NULL)
         {
         fprintf(ausgabe_file,"\n********** ERROR **********\n",error_str);
         fprintf(ausgabe_file,"error=vertices  %s \n",error_str);
         fclose(ausgabe_file);
         }

   exit(0);
   }

int ORT::laden(char* filename)
   {
   char ort_str[100],*endptr,str[10];
   int i=1,ort_nr;
    double help;
   FILE *file;
   if ((file=fopen(filename,"r"))!=NULL) {

      fgets(ort_str,100,file);
      while (!feof(file))
         {
         if(fgets(ort_str,100,file)!=NULL)
            ORT::anzahl++;
         }
      if(!(ort = new ORT[ORT::anzahl+1]))
         return -9;//ort->error(-9,"c:loc_error");
      fseek(file,0,SEEK_SET);
      fgets(ort_str,100,file);
      //gets(str);
      ORT::ges_gewicht=0;
      for(i=1;i<=ORT::anzahl;i++)
         {
         fgets(ort_str,100,file);

         if(ort_str != NULL)
            {
            ort_nr=atoi(strtok(ort_str,","));
            if((ort_nr>ORT::anzahl)||(ort_nr<1))
               return -2;

   ort[ort_nr].gewicht=strtod(strtok(0,","),&endptr);//strtod(strtok(ort_str,",")
,&endptr);
            ORT::ges_gewicht+=ort[ort_nr].gewicht;
            //printf(" nach gewicht ort_nr = %d \n",ort_nr);
            ort[ort_nr].kapunter=0;
            ort[ort_nr].kapober=FLT_MAX;
            //ort[ort_nr].kapunter=strtod(strtok(0,","),&endptr);
            //ort[ort_nr].kapober=strtod(strtok(0,",\n"),&endptr);
            ort[ort_nr].zuge=0;
            ort[ort_nr].vorge=0;
            ort[ort_nr].ausge=0;
            ort[ort_nr].anfa=0;
            }
         }
      fclose(file);
      return 1;    }
   else
      return -1;

   }

void ORT::zeigen()
   {
   int i;
    char str[5];
   for(i=1;i<=ort->anzahl;i++)
```

```
   {
      printf(" %d.Ort Gew. %f vorge %d  ausge %d zuge %d\n"      //Kap.ober %f
Kap.unter.
     %f,i,ort[i].gewicht,ort[i].vorge,ort[i].ausge,ort[i].zuge) ;   if(i==20)
        getch();
      }
   }
void ORT::error(int error_nr,char *ausgabe)
   {
   char error_str[100];
   FILE *ausgabe_file;
   switch(error_nr)
      {
      case -1:
         strcpy(error_str,"file with nodes not found");
         break;
      case -2:
         strcpy(error_str," nodes out of range");
         break;
      case -9:
         strcpy(error_str,"Out of memory1");

         break;
      case -10:
         strcpy(error_str,"Out of memory2");

         break;
      case -11:
         strcpy(error_str,"Out of memory3");

         break;
      }
      if ((ausgabe_file=fopen(ausgabe,"a"))!=NULL)
         {
         fprintf(ausgabe_file,"\n********** ERROR **********\n",error_str);
         fprintf(ausgabe_file,"error=nodes : %s \n",error_str);
         fclose(ausgabe_file);
         }
   exit(0);
   }
int MODELL::working()
   {
   printf("working...\n");
   return 0;
  }
int MODELL::laden(char* ortsfile,char* distanzfile,char* modellfile,char*
ausgabefile,char* regfile)
   {
   int i,v,ac,vc;
   char mod_str[500],*endptr,varname[30],str[3],*vor;
   FILE * modell_file,* ausgabe_file;
   if ((modell_file=fopen(modellfile,"r"))!=NULL)
      {
      modell.zielfunktion=0;
      modell.kapober=-1;
      modell.kapunter=0;
      modell.reg_anz=-1;
      modell.maxdis=-1;
      modell.a1=-1;
      modell.c1=-1;
      modell.aexp=0;
      modell.bexp=0;
      modell.g=0;
      modell.versuche=1;
      modell.versuche_sol=1;
      ac=0;
      vc=0;
      while (!feof(modell_file))
           {
```

```
if(fgets(mod_str,500,modell_file)==NULL)
   break;
strncpy(varname,strtok(mod_str,"="),30);
if(strcmp(varname,"max_capacity")==0)
   {
   modell.kapober=strtod(strtok(0,",\n"),&endptr);
   for(i=1;i<=ORT::anzahl;i++)
      ort[i].kapober=modell.kapober;
   }
if(strcmp(varname,"min_capacity")==0)
   {
   modell.kapunter=strtod(strtok(0,",\n"),&endptr);
   for(i=1;i<=ORT::anzahl;i++)
      ort[i].kapunter=modell.kapunter;
   }
if(strcmp(varname,"trial_optimize")==0)
   modell.versuche=atoi(strtok(0,",\n"));
if(strcmp(varname,"trial_solution")==0)
   modell.versuche_sol=atoi(strtok(0,",\n"));
if(strcmp(varname,"objective")==0)
   modell.zielfunktion=atoi(strtok(0,",\n"));
if(strcmp(varname,"maxdis")==0)
   modell.maxdis=strtod(strtok(0,",\n"),&endptr);
if(strcmp(varname,"a1")==0)
   modell.a1=strtod(strtok(0,",\n"),&endptr);
if(strcmp(varname,"c1")==0)
   modell.c1=strtod(strtok(0,",\n"),&endptr);
if(strcmp(varname,"a_exp")==0)
   modell.aexp=strtod(strtok(0,",\n"),&endptr);
if(strcmp(varname,"b_exp")==0)
   modell.bexp=strtod(strtok(0,",\n"),&endptr);
if(strcmp(varname,"g_dist")==0)
   modell.g=strtod(strtok(0,",\n"),&endptr);

if(strcmp(varname,"number_of_regions")==0)
{
   modell.reg_anz=atoi(strtok(0,",\n"));
   modell.vor_geg=new int[modell.reg_anz+1];
   modell.anfang=new int[modell.reg_anz+1];
   for(ac=0;ac<=modell.reg_anz;ac++)
      {
      modell.vor_geg[ac]=-1;
      modell.anfang[ac]=-1;
  }
ac=0;
}
if(strcmp(varname,"fixed")==0)
if(modell.reg_anz!=-1)
   {

   while((vor=strtok(NULL,",\n"))!=NULL)
      {
      v=atoi(vor);
      if((v>=1)&&(v<=ORT::anzahl) && ((vc+ac)<=modell.reg_anz))
         {
         if(ort[v].ausge!=1)  //wenn ort nicht ausgeschlossen ist
            {
            ort[v].vorge=1;
            vc++;
            modell.vor_geg[vc]=v;
            }
         }
      }
   }
if(strcmp(varname,"excluded")==0)
   while((vor=strtok(NULL,",\n"))!=NULL)
      {
      v=atoi(vor);
      if((v>=1)&&(v<=ORT::anzahl))
```

```
               if((ort[v].vorge!=1) && (ort[v].anfa!=1))
                  ort[v].ausge=1;
            }
         if(strcmp(varname,"start")==0)
         if(modell.reg_anz!=-1)
            {
            while((vor=strtok(NULL,",\n"))!=NULL)
               {
               v=atoi(vor);
               if((v>=1)&&(v<=ORT::anzahl) && ((vc+ac)<=modell.reg_anz))
                  {
                  if(ort[v].ausge!=1)  //wenn ort nicht ausgeschlossen ist
                     {
                     ort[v].anfa=1;
                     ac++;
                     modell.anfang[ac]=v;
                      }
                  }
               }
            }
      fclose(modell_file);
      }
   else
      {
      modell.error(-1,modellfile);
      }
   if(modell.reg_anz==-1)
      modell.error(-2,modellfile);

   fprintf(ausgabe_file,"Modelldaten,Regionsanzahl:,%d,Kap.ober:,%f,Kap.unter:,%f
\n",modell.reg_anz,modell.kapober,modell.kapunter);
   if(modell.maxdis==-1)
      modell.maxdis=FLT_MAX;
   if(modell.kapober==-1)
      modell.kapober=FLT_MAX;
   return 1;
   //gets(str);
   }
void MODELL::zeigen()
   {
    printf("Regionsanzahl : %d \n maxdis= %f a1= %f c1= %f \n aexp= %f bexp= %f
g= %f\n"

,modell.reg_anz,modell.maxdis,modell.a1,modell.c1,modell.aexp,modell.bexp,modell.
g);
    }


// Hauptprogramm

void main(int argc, char *argv[])
{
int i,gef_ort,gef_reg,test,error_nr=1,error_nr_a3=0,counter_alba2=0,
   komplett,versuch,best=-1,*best_anfang,r,a3_zf,best_a3;
float dum_dist,dist_zent=0,zf_glo_zw,zf_glo=FLT_MAX;
char str[5],verb_str[200],verb_kom_str[200]="";
modell.working();
if(argc==4)
   {
   ORT::anzahl=atoi(argv[3]);
   strcpy(verb_str,argv[1]);
   strcpy(verb_kom_str,argv[2]);
   }
else
   {

   dum_dist=0;
   strcpy(verb_str,argv[2]);
```

```
   if(argc==6)
      strcpy(verb_kom_str,argv[5]);
   if((error_nr=ort->laden(argv[1]))<0)
        ort->error(error_nr,argv[3]);
   modell.working();
   if((error_nr=modell.laden(argv[1],argv[2],argv[3],"loc_error",argv[4]))<0)
      modell.error(error_nr,argv[3]);
   if((error_nr=region->initialisieren(modell.reg_anz))<1)
      region->error(error_nr,argv[3]);
   modell.working();
   }
if((error_nr=verbindung->laden(verb_str))<0)
   {
   verbindung->error(error_nr,argv[3]);
   }
modell.working();
if(error_nr==1)
   komplett=1;
else
   komplett=-1;
if(argc==6)
   {
   komplett=verbindung->kom_laden(verb_kom_str);
   }
modell.working();
if(komplett==-1)
   if((error_nr=verbindung->komplett())<0)
     {
      //verbindung->zeigen();
      verbindung->error(error_nr,argv[3]);
      }
modell.working();
if((argc>5)&&(komplett==-1))
   verbindung->kom_speichern(verb_kom_str);
if(argc!=4)
{
counter_alba2=0;
error_nr=-5;
best=-1;
a3_zf=0;
best_a3=-1;
best_anfang=new int[modell.reg_anz+1];
while((counter_alba2<modell.versuche_sol)&&(a3_zf<modell.versuche)) //
   {
   counter_alba2++;
   for(versuch=1;versuch<=2;versuch++)
      {
         region->orte_loeschen();
         error_nr=region->alba2(versuch);
         modell.working();
         if(error_nr>0)
            {
            if((error_nr_a3=region->alba3(0))>0)//modell.zielfunktion
               {
               modell.working();
           counter_alba2=0;
               a3_zf++;
               zf_glo_zw=region->zielfunc_glo(modell.zielfunktion);
               if(zf_glo_zw < zf_glo)
                  {
                  zf_glo=zf_glo_zw;
                  best=versuch;
                  best_a3=a3_zf;
                  region-
>in_file_schreiben(argv[3],argv[4],counter_alba2,modell.anfang,best);
                  for(r=1;r<=region->anzahl;r++)
               best_anfang[r]=modell.anfang[r];
                  }  //if best
               }   //if error_nr_a3(alba3)
```

```
          }   //if error_nr(alba2)
      }//for
   } //while
if(best==-1)
      {
      region->error(error_nr,argv[3]);
       }
}
else
   {
   printf("vor verb komplett speichern \n");
   verbindung->kom_speichern(verb_kom_str);
   printf(" nach verb komplett speichern\n");
  }
}
```

Anhang C: Programmcode der graphischen Benutzerschnittstelle GUI-LOC-ALLOC (Avenue-Programmcode)

Die Realisierung der graphischen Benutzerschnittstelle erfolgte über die Programmiersprache **Avenue.** Dies ist die in ArcView zur Verfügung stehende Programmierumgebung. Die Benutzerschnittstelle besteht aus vielen kleinen Programmen (scripts), von denen einige automatisch beim Starten des Projektes `gui_loc_alloc.apr` ausgeführt werden, die meisten aber durch eine Menüauswahl des Benutzers gestartet werden. Dementsprechend sind hier zuerst jene scripts angeführt, die beim Start des Projektes zur Ausführung gelangen und im folgenden die den jeweiligen Menüoptionen zugeordneten.

script „`alloc_start`“: Wird beim Starten des Projektes ausgeführt und ruft „*`alloc_gui`*“ und „*`initialise`*“ auf.

```
av.Run("alloc_gui",{})
av.Run("initialise",{})
if(_interface="Loc Alloc") then
  user_view=view.Make
  user_view_win=user_view.GetWin
  user_view_win.Open
  user_view_win.Maximize
  user_view_win.Activate

end
```

script „`alloc_gui`“: Definiert die Benutzerschnittstelle.

```
interface_liste={"Loc Alloc","Arc View with Loc Alloc"}
_interface=MsgBox.choiceAsString(interface_liste,"Choose a Graphic User
Interface","Loc-Alloc")
if(_interface=nil) then
  exit
end
av.GetProject.ResetGUIs
alloc_GUI=av.FindGUI("View")
alloc_MenuBar= alloc_GUI.GetMenuBar
alloc_ButtonBar= alloc_GUI.GetButtonBar
alloc_ToolBar= alloc_GUI.GetToolBar
chart_GUI=av.FindGUI("Chart")
chart_MenuBar= chart_GUI.GetMenuBar
chart_ButtonBar= chart_GUI.GetButtonBar
chart_ToolBar= chart_GUI.GetToolBar

n=NameDictionary.Make(300)
for each i in IconMgr.GetIcons
  n.Add(i)
```

```
end

if(_interface="Loc Alloc") then

'****************Umbenennen des User Interfaces / Löschen aller anderen
Funktionen ************
av.SetName("LOC-ALLOC")
alloc_GUI.SetDocBaseName("Locations")
alloc_GUI.SetTitle("Loc-Alloc")
alloc_GUI.SetIcon(n.Get("Home"))

av.FindGUI("Table").SetVisible(false)
av.FindGUI("Chart").SetVisible(false)
av.FindGUI("Script").SetVisible(false)
av.FindGUI("Layout").SetVisible(false)
alloc_MenuBar.Empty
alloc_ButtonBar.Empty
old_tool=alloc_ToolBar.GetControls.Get(12)
alloc_ToolBar.Remove(old_tool)
old_tool=alloc_ToolBar.GetControls.Get(11)
alloc_ToolBar.Remove(old_tool)
old_tool=alloc_ToolBar.GetControls.Get(9)
alloc_ToolBar.Remove(old_tool)
old_tool=alloc_ToolBar.GetControls.Get(8)
alloc_ToolBar.Remove(old_tool)
old_tool=alloc_ToolBar.GetControls.Get(2)
alloc_ToolBar.Remove(old_tool)

'chart_MenuBar.Empty
'chart_ButtonBar.Empty
'chart_ToolBar.Empty
'for each con in alloc_MenuBar.GetControls
  'alloc_MenuBar.Remove(con)
'end
'for each con in alloc_ButtonBar.GetControls
'  alloc_ButtonBar.Remove(con)
'end
end

'******************  Creating LocAlloc Interface **********************
n=NameDictionary.Make(300)
for each i in IconMgr.GetIcons
  n.Add(i)
end

'********************** Create Tools ***********************************
fix_tool=Tool.Make
fix_tool.SetIcon(n.Get("Hut"))
fix_tool.SetClick("set_fix")
fix_tool.SetHelp("Specify set of fixed supply locations ")

excl_tool=Tool.Make
excl_tool.SetIcon(n.Get("Cut"))
excl_tool.SetClick("set_excl")
excl_tool.SetHelp("Specify set of excluded supply locations ")

start_tool=Tool.Make
start_tool.SetIcon(n.Get("Run"))
start_tool.SetClick("set_start")
start_tool.SetHelp("Specify set of excluded supply locations ")
sel_loc_tool=Tool.Make
sel_loc_tool.SetIcon(n.Get("SelectAll"))
```

```
sel_loc_tool.SetClick("sel_loc")
sel_loc_tool.SetHelp("Enter selected locations as set of specified ")

clear_loc_tool=Tool.Make
clear_loc_tool.SetIcon(n.Get("SelectNone"))
clear_loc_tool.SetClick("clear_loc")
clear_loc_tool.SetHelp("Clear set of specified locations )

show_loc_tool=Tool.Make
show_loc_tool.SetIcon(n.Get("Frame"))
show_loc_tool.SetClick("show_loc")
show_loc_tool.SetHelp("Show specification ")

'************************ Make Tool Menu Bar ***************************
loc_ToolMenu=toolMenu.Make
loc_ToolMenu_con= loc_ToolMenu.GetControls
loc_ToolMenu_con.Add(fix_tool)
loc_ToolMenu_con.Add(excl_tool)
loc_ToolMenu_con.Add(start_tool)

alloc_ToolBar.Add(loc_ToolMenu,25)
alloc_ToolBar.Add(show_loc_tool,25)
alloc_ToolBar.Add(sel_loc_tool,25)
alloc_ToolBar.Add(clear_loc_tool,25)

'************************ Make Buttons  ****************************
print_view_but=Button.Make
print_view_but.SetIcon(n.Get("Print"))
print_view_but.SetClick("view.Print")
print_view_but.SetHelp("Print ")

print_but=Button.Make
print_but.SetIcon(n.Get("Print"))
print_but.SetClick("doc.Print")
print_but.SetHelp("Print ")

'************************ Make Button Bar ****************************
alloc_ButtonBar.Add(print_view_but,30)
chart_ButtonBar.Add(print_but,30)

'*************************** Create Menus ******************************
'************************ Files Menu ***********************************
graph_Choice =Choice.Make
graph_Choice.SetLabel("Graph Specification")
graph_Choice.SetClick("graph_spec")
graph_rep_Choice =Choice.Make
graph_rep_Choice.SetLabel("Graph Representation ")
graph_rep_Choice.SetClick("graph_rep")

files_Menu=Menu.Make
files_Menu.SetLabel("Graph Files")
files_Menu_con= files_Menu.GetControls
files_Menu_con.Add(graph_Choice)
files_Menu_con.Add(graph_rep_Choice)

'***************************** Location Criteria Menue ******************
ini_Choice =Choice.Make
ini_Choice.SetLabel("Initialise")
ini_Choice.SetClick("initialise")
con_Choice =Choice.Make
con_Choice.SetLabel("Constraints")
con_Choice.SetClick("constraints")
```

```
obj_Choice =Choice.Make
obj_Choice.SetLabel("Objectives")
obj_Choice.SetClick("objectives")
alg_Choice =Choice.Make
alg_Choice.SetLabel("Algorithm Parameters")
alg_Choice.SetClick("algo_par")
sav_Choice =Choice.Make
sav_Choice.SetLabel("Save Specification")
sav_Choice.SetClick("save_parameter")
loa_Choice =Choice.Make
loa_Choice.SetLabel("Load Specification")
loa_Choice.SetClick("load_parameter")

loccrit_Menu=Menu.Make
loccrit_Menu.SetLabel("Location Criteria")
loccrit_Menu_con= loccrit_Menu.GetControls
loccrit_Menu_con.Add(ini_Choice)
loccrit_Menu_con.Add(con_Choice)
loccrit_Menu_con.Add(obj_Choice)
loccrit_Menu_con.Add(alg_Choice)
loccrit_Menu_con.Add(sav_Choice)
loccrit_Menu_con.Add(loa_Choice)

'***************************  Solve Menu  ***************************
sol_Choice =Choice.Make
sol_Choice.SetLabel("Start Computation")
sol_Choice.SetClick("locate")

solve_Menu=Menu.Make
solve_Menu.SetLabel("Solve")
solve_Menu_con= solve_Menu.GetControls
solve_Menu_con.Add(sol_Choice)

'************************** Display Menu ******************************
sel_Choice =Choice.Make
sel_Choice.SetLabel("Select Solution File")
sel_Choice.SetClick("sel_reg_tab")
reg_Choice =Choice.Make
reg_Choice.SetLabel("Display Solution")
reg_Choice.SetClick("show_sol")
sta_Choice =Choice.Make
sta_Choice.SetLabel("Display Statistics for Solution")
sta_Choice.SetClick("chart_crea")
cle_Choice =Choice.Make
cle_Choice.SetLabel("Clear Solution")
cle_Choice.SetClick("clear_reg")
chi_Choice =Choice.Make
chi_Choice.SetLabel("Change Interface")
chi_Choice.SetClick("alloc_gui")

disp_Menu=Menu.Make
disp_Menu.SetLabel("Display Solution")
disp_Menu_con= disp_Menu.GetControls
disp_Menu_con.Add(sel_Choice)
disp_Menu_con.Add(reg_Choice)
disp_Menu_con.Add(sta_Choice)
disp_Menu_con.Add(cle_Choice)
disp_Menu_con.Add(chi_Choice)

'************************ Make Menu Bar ******************************
alloc_MenuBar.Add(files_Menu,15)
alloc_MenuBar.Add(loccrit_Menu,15)
alloc_MenuBar.Add(solve_Menu,15)
```

```
alloc_MenuBar.Add(disp_Menu,15)
```

script „Initialise“: Initialisiert die globalen Variablen.

```
'**************************initialise***********************
_loc_kind=1
_reg_mod_fn="mod.txt".AsFileName
_nodes=""
_nodes_neighbor=""
_vertices=""
_nodes_shp_fn="".AsFileName
_vertices_shp_fn="".AsFileName
_reg_shp_fn="".AsFileName
_reg_tab_fn="reg.txt".AsFileName
_cent_leg_fn="".AsFileName
_reg_leg_fn="".AsFileName
_maxdis="-1"
_c1="-1"
_a1="-1"
_kapunter="-1"
_kapober="-1"
_reg_anz="1"
_vorgegeben="-1"
_ausgeschlossen="-1"
_anfang="-1"
_zielfunktion="0"
_aexp="1"
_bexp="1"
_g="0"
_versuche_opt="1"
_versuche_sol="1"
```

Im weiteren wird anhand der **Menüpunkte** und ihrer *Optionen* vorgegangen:

Für den Menüpunkt **Files** , Option *Graph*:

script „graph“: Lädt die Dateinamen der Textdateien des Graphen.

```
_nodes=FileDialog.Show("*.txt","text file","Nodes graph
file").GetBaseName
if(_nodes=nil) then
  exit
end
_vertices =FileDialog.Show("*.txt"," text file","Vertices(shorted
distances) graph file").GetBaseName
if(_vertices=nil) then
  exit
end
_nodes_neighbor=FileDialog.Show("*.txt"," text file"," nodes neighbor
graph file").GetBaseName
if(_nodes_neighbor=nil) then
  exit
end
```

Für den Menüpunkt **Files** , Option *Graph Representation:*

script „graph_rep“: Lädt die Dateinamen der Graphikdateien des Graphen.
und stellt ihn dar

```
_nodes_shp_fn =FileDialog.Show("*.shp","shape File","Node shape file")
if(_nodes_shp_fn=nil) then
  exit
end
_vertices_shp_fn =FileDialog.Show("*.shp","shape File","Vertices shape
file")
if(_nodes_shp_fn=nil) then
  exit
end
_reg_shp_fn =FileDialog.Show("*.shp","shape File","Regions shape file")
if(_reg_shp_fn=nil) then
  exit
end
_cent_leg_fn =FileDialog.Show("*.avl","legend File","Center legend file")
if(_cent_leg_fn=nil) then
  exit
end
_reg_leg_fn =FileDialog.Show("*.avl","legend File","Region legend file")
if(_reg_leg_fn=nil) then
  exit
end
theView=av.GetActiveDoc
theThemeList=theView.GetThemes
for each entry in theThemeList
  if((_nodes_shp_fn.GetBaseName=entry.AsString)or(_reg_shp_fn.GetBaseNam
e=entry.AsString)) then
    exit
  end
end
if((_nodes_shp_fn=nil)or(_reg_shp_fn=nil)) then
  MsgBox.Error("No file names for graphic representation
available","ERROR")
  exit
end

'******** create nodes theme    *************
nodes_SrcName=SrcName.Make(_nodes_shp_fn.AsString)
nodes_theme=Theme.Make(nodes_SrcName)
if(nodes_theme=nil) then
  MsgBox.Error("No graphic representation available: Center
representation theme couldnt be build","Error")
  exit
end
nodes_theme.SetName(_nodes_shp_fn.GetBaseName)

'******** create vertices theme ********
vertices_SrcName=SrcName.Make(_vertices_shp_fn.AsString)
vertices_theme=Theme.Make(vertices_SrcName)
if(vertices_theme=nil) then
  MsgBox.Error("No graphic representation available: vertices
representation theme couldnt be build","Error")
end
vertices_theme.SetName(_vertices_shp_fn.GetBaseName)

'******** create regions theme********
reg_SrcName=SrcName.Make(_reg_shp_fn.AsString)
reg_theme=Theme.Make(reg_SrcName)
if(reg_theme=nil) then
  MsgBox.Error("No graphic representation available: Regions
representation theme couldnt be build","Error")
  exit
end
```

```
reg_theme.SetName(_reg_shp_fn.GetBaseName)

theView.AddTheme(reg_theme)
theView.AddTheme(nodes_theme)
theView.AddTheme(vertices_theme)
nodes_theme.SetVisible(true)
reg_theme.SetVisible(true)
vertices_theme.SetVisible(true)
nodes_theme.SetActive(true)
```

Für den Menüpunkt **Location Criteria** , Option *Initialise:*

script „Initialise“: Initialisiert die globalen Variablen, siehe oben.

Für den Menüpunkt **Location Criteria** , Option *Constraints:*

script „Constraints“: Dialogfeld für Randbedingungen.

```
default=List.Make
default.Add(_reg_anz)
default.Add(_vorgegeben)
default.Add(_ausgeschlossen)
default.Add(_maxdis)
default.Add(_c1)
default.Add(_a1)
default.Add(_kapunter)
default.Add(_kapober)
EntryList = {"Number of  Regions ","Fixed Locations","Excluded
Locations","Maximum distance","Second distance(c1)","maximum weight
beyond c1(a1)","General minimum capacity","General maximum capacity"}
modellinfo = MsgBox.MultiInput("Parameter
Values","Constraints",EntryList,default)
if(modellinfo.Count <> 0) then
  _reg_anz = modellinfo.Get(0)
  _vorgegeben=modellinfo.Get(1)
  _ausgeschlossen =modellinfo.Get(2)
  _maxdis=modellinfo.Get(3)
  _c1=modellinfo.Get(4)
  _a1=modellinfo.Get(5)
  _kapunter=modellinfo.Get(6)
  _kapober=modellinfo.Get(7)
end
```

Für den Menüpunkt **Location Criteria** , Option *Objectives:*

script „Objectives“: Dialogfeld für Zielfunktionen.

```
default=List.Make
default.Add("Minimize distance")
default.Add("Minimize Maximum distance")
default.Add("Minimize Weight beyond Second distance(c1)")
default.Add("P-Median Problem")
EntryList=List.Make
EntryList.Add(default.Get(_zielfunktion.AsNumber))
for each i in 0..3
  if(i<>_zielfunktion.AsNumber)  then
    EntryList.Add(default.Get(i))
  end
```

```
end
modellinfo = MsgBox.ChoiceAsString(EntryList,"objective functions","LOC-
ALLOC")
if(modellinfo="Minimize distance") then
  _zielfunktion="0"
end
if(modellinfo="Minimize Maximum distance") then
  _zielfunktion="1"
end
if(modellinfo="Minimize Demand beyond Second distance(c1)") then
  _zielfunktion="2"
end
if(modellinfo="Minimize weighted distance") then
  _zielfunktion="3"
end
default1=List.Make
default1.Add(_aexp)
default1.Add(_bexp)
default1.Add(_g)
default1.Add(_c1)
EntryList = {"aexp","bexp","g","Second distance(c1)"}
modellinfo = MsgBox.MultiInput("objectives","LOC-
ALLOC",EntryList,default1)
if(modellinfo.Count <> 0) then
  _aexp=modellinfo.Get(0)
  _bexp=modellinfo.Get(1)
  _g=modellinfo.Get(2)
  _c1=modellinfo.Get(3)
end
```

Für den Menüpunkt **`Location Criteria`** , Option ***`Algorithem Parameters:`***

script „`Algo_Para`“: Dialogfeld für Parameter des Algorithmus.

```
default=List.Make
default.Add(_anfang)
default.Add(_versuche_opt)
default.Add(_versuche_sol)
EntryList = {"starting nodes ","number of trials for optimization","
number of trials for finding a solution "}
modellinfo = MsgBox.MultiInput("algorithm parameters","LOC-
ALLOC",EntryList,default)
if(modellinfo.Count <> 0) then
  _anfang =modellinfo.Get(0)
  _versuche_opt=modellinfo.Get(1)
  _versuche_sol=modellinfo.Get(2)
end
```

Für den Menüpunkt **`Location Criteria`** , Option ***`Save Specification:`***

script „`save_para`“: Speicherung der Parameter in Datei.

```
_reg_mod_fn=FileDialog.Put("*.txt".AsFileName,"*.txt","Save Parameter
File")
if(_reg_mod_fn= nil) then
   exit
end
mod_fi=LineFile.Make(_reg_mod_fn,#FILE_PERM_WRITE)
mod_fi.WriteElt("********** graph files **********")
mod_fi.WriteElt("nodes="+_nodes)
```

```
mod_fi.WriteElt("vertices="+_vertices)
mod_fi.WriteElt("nodes_neighbor="+_nodes_neighbor)
mod_fi.WriteElt("********** constraints **********")
mod_fi.WriteElt("number_of_regions="+_reg_anz)
if(_vorgegeben<>"-1") then
  mod_fi.WriteElt("fixed="+_vorgegeben)
end
if(_ausgeschlossen<>"-1") then
  mod_fi.WriteElt("excluded="+_ausgeschlossen)
end
if(_maxdis<>"-1") then
  mod_fi.WriteElt("maxdis="+_maxdis)
end
mod_fi.WriteElt("c1="+_c1)
if(_a1<>"-1") then
  mod_fi.WriteElt("a1="+_a1)
end
if(_kapunter<>"-1") then
  mod_fi.WriteElt("min_capacity="+_kapunter)
end
if(_kapober<>"-1") then
  mod_fi.WriteElt("max_capacity="+_kapober)
end
mod_fi.WriteElt("********** objective function **********")
mod_fi.WriteElt("objective="+_zielfunktion)
mod_fi.WriteElt("a_exp="+_aexp)
mod_fi.WriteElt("b_exp="+_bexp)
mod_fi.WriteElt("g_dist="+_g)
mod_fi.WriteElt("********** algorithm parameters **********")
if(_anfang<>"-1") then
  mod_fi.WriteElt("start="+_anfang)
end
mod_fi.WriteElt("trial_optimize="+_versuche_opt)
mod_fi.WriteElt("trial_solution="+_versuche_sol)
mod_fi.WriteElt("********** graph  arc view representation **********")
mod_fi.WriteElt("nodes_shp="+_nodes_shp_fn.AsString)
mod_fi.WriteElt("vertices_shp="+_vertices_shp_fn.AsString)
mod_fi.WriteElt("reg_shp="+_reg_shp_fn.AsString)
mod_fi.WriteElt("cent_leg="+_cent_leg_fn.AsString)
mod_fi.WriteElt("reg_leg="+_reg_leg_fn.AsString)
mod_fi.Close
```

Für den Menüpunkt **`Location Criteria`** , Option ***`Load Specification:`***

script „`load_para`“: Laden der Parameter aus Datei und Darstellung des Modells.

```
_reg_mod_fn=FileDialog.Show("*.txt","Text File","Parameter File")
if(_reg_mod_fn=nil) then
   exit
end
mod_fi=TextFile.Make(_reg_mod_fn,#FILE_PERM_READ)
if(mod_fi=nil) then
   MsgBox.Warning("Cannot open file: "++_reg_mod_fn.AsString,"ERROR")
   exit
end
reg_mod_str= mod_fi.Read(mod_fi.GetSize)
MsgBox.Report(reg_mod_str,"Actual Parameters")
actual=MsgBox.YesNo("Do you want to use these Parameters","Check
Actuality",true)
IF(not(actual)) then
   exit
```

```
end
mod_fi.Close
mod_fi=LineFile.Make(_reg_mod_fn,#FILE_PERM_READ)
if(mod_fi=nil) then
   MsgBox.Warning("File for Parameters defined does not exist","ERROR")
   exit
end
parameter=List.Make
  while (true)
    str=mod_fi.ReadElt
    if(str=nil) then
      break
    else
      parameter.Add(str)
    end
  end
  mod_fi.Close
  for each element in parameter
    if ( element.contains("number_of_regions=")) then
      _reg_anz=element.Substitute("number_of_regions=","")
    end
    if ( element.contains("fixed=")) then
      _vorgegeben=element.Substitute("fixed=","")
    end
    if ( element.contains("excluded=")) then
      _ausgeschlossen=element.Substitute("excluded=","")
    end
    if ( element.contains("start=")) then
       _anfang=element.Substitute("start=","")
    end
    if ( element.contains("maxdis=")) then
      _maxdis= element.Substitute("maxdis=","")
    end
    if ( element.contains("c1=")) then
       _c1=element.Substitute("c1=","")
    end
    if ( element.contains("a1=")) then
      _a1=element.Substitute("a1=","")
    end
    if ( element.contains("min_capacity=")) then
      _kapunter=element.Substitute("min_capacity=","")
    end
    if ( element.contains("max_capacity=")) then
      _kapober=element.Substitute("max_capacity=","")
    end
    if ( element.contains("objective=")) then
      _zielfunktion=element.Substitute("objective=","")
    end
    if ( element.contains("a_exp=")) then
      _aexp=element.Substitute("a_exp=","")
    end
    if ( element.contains("b_exp=")) then
      _bexp=element.Substitute("b_exp=","")
    end
    if ( element.contains("g_dist=")) then
      _g=element.Substitute("g_dist=","")
    end
    if ( element.contains("trial_solution=")) then
      _versuche_sol=element.Substitute("trial_solution=","")
    end
   if ( element.contains("trial_optimize=")) then
      _versuche_opt=element.Substitute("trial_optimize=","")
    end
```

```
if ( element.contains("nodes_shp=")) then
       _nodes_shp_fn=element.Substitute("nodes_shp=","").AsFileName
    end
if ( element.contains("vertices_shp=")) then
       _vertices_shp_fn=element.Substitute("vertices_shp=","").AsFileName
    end
if ( element.contains("reg_shp=")) then
      _reg_shp_fn=element.Substitute("reg_shp=","").AsFileName
    end
if ( element.contains("cent_leg=")) then
      _cent_leg_fn=element.Substitute("cent_leg=","").AsFileName
    end
if ( element.contains("reg_leg=")) then
      _reg_leg_fn=element.Substitute("reg_leg=","").AsFileName
    end
if ( element.contains("nodes=")) then
      _nodes=element.Substitute("nodes=","")
    end
if ( element.contains("vertices=")) then
      _vertices=element.Substitute("vertices=","")
    end
if ( element.contains("nodes_neighbor=")) then
      _nodes_neighbor=element.Substitute("nodes_neighbor=","")
    end
 end
theView=av.GetActiveDoc
theThemeList=theView.GetThemes
for each entry in theThemeList

  if((_nodes_shp_fn.GetBaseName=entry.AsString)or(_reg_shp_fn.GetBaseNam
e=entry.AsString)) then
     exit
  end
end
if((_nodes_shp_fn=nil)or(_reg_shp_fn=nil)) then
  MsgBox.Error("No file names for graphic representation
available","ERROR")
  exit
end

'******** create nodes theme ********
nodes_SrcName=SrcName.Make(_nodes_shp_fn.AsString)
nodes_theme=Theme.Make(nodes_SrcName)
if(nodes_theme=nil) then
  MsgBox.Error("No graphic representation available: Center
representation theme couldnt be build","Error")
  exit
end
nodes_theme.SetName(_nodes_shp_fn.GetBaseName)

'******** create vertices theme ********
vertices_SrcName=SrcName.Make(_vertices_shp_fn.AsString)
vertices_theme=Theme.Make(vertices_SrcName)
if(vertices_theme=nil) then
  MsgBox.Error("No graphic representation available: vertices
representation theme couldnt be build","Error")
end
vertices_theme.SetName(_vertices_shp_fn.GetBaseName)

'******** create regions theme********
reg_SrcName=SrcName.Make(_reg_shp_fn.AsString)
reg_theme=Theme.Make(reg_SrcName)
if(reg_theme=nil) then
```

```
   MsgBox.Error("No graphic representation available: Regions
representation theme couldnt be build","Error")
   exit
end
reg_theme.SetName(_reg_shp_fn.GetBaseName)
theView.GetWin.Maximize
theView.AddTheme(reg_theme)
theView.AddTheme(nodes_theme)
theView.AddTheme(vertices_theme)
nodes_theme.SetVisible(true)
'reg_theme.SetVisible(true)
vertices_theme.SetVisible(true)

nodes_theme.SetActive(true)

'*****************load  nodes weight legend*****************
cent_shp_leg=nodes_theme.GetLegend
cent_shp_leg.load(_cent_leg_fn,#LEGEND_LOADTYPE_ALL)
nodes_theme.UpdateLegend

'************** set view to size of themes********************
av.GetProject.SetModified(true)
theView = av.GetActiveDoc
theThemes = theView.GetActiveThemes
r = Rect.MakeEmpty
for each t in theThemes
  r = r.UnionWith(t.ReturnExtent)
end
if (r.IsEmpty) then
  return nil
elseif ( r.ReturnSize = (0@0) ) then
  theView.GetDisplay.PanTo(r.ReturnOrigin)
else
  theView.GetDisplay.SetExtent(r.Scale(1.1))
end

'*****************labeling  nodes and vertices***************
nodes_shp_ftab= nodes_theme.GetFtab
ort_nr_nodes_shp_f=nodes_shp_ftab.FindField("ort_nr")
aTextSym = TextSymbol.Make
aTextSym.SetFont(Font.Make("Times New Roman","Standard"))
aTextSym.SetSize(3.2)
nodes_theme.GetTextPositioner.SetHAlign(#TEXTPOSITIONER_HALIGN_CENTER)
nodes_theme.GetTextPositioner.SetVAlign(#TEXTPOSITIONER_VALIGN_ON)
if(nodes_theme.GetLabelField=nil) then
   nodes_theme.SetLabelField(ort_nr_nodes_shp_f)
  aTextSym.SetFont(Font.Make("Times New Roman","Bold"))
  aTextSym.SetSize(10)
end
nodes_theme.SetFixedSizeText(false)
nodes_theme.SetLabelTextSym(aTextSym)
aLabeler=Labeler.Make(av.GetActiveDoc.GetDisplay.ReturnVisExtent)
aLabeler.SetLinePlacement(0)
aLabeler.SetFeatureWeight(#LABEL_WEIGHT_NO)
aLabeler.SetLabelWeight(#LABEL_WEIGHT_HIGH)
aLabeler.Load(nodes_theme)
if(nodes_theme.GetLabelField=ort_nr_nodes_shp_f) then
   av.GetActiveDoc.GetAutoLabels(aLabeler,false)
else
  av.GetSymbolWin.Open
  AutoLabelDialog.Show(nodes_theme)
end
exit
```

```
vertices_theme.SetActive(true)
' MsgBox.Info ("vor vert label", "")
vertices_shp_ftab= vertices_theme.GetFtab
dist_vertices_shp_f=vertices_shp_ftab.FindField("Distanz")
if(dist_vertices_shp_f=nil) then
  MsgBox.Info ("no distanz field", "")
    exit
end
if(vertices_theme.GetLabelField=nil) then
  vertices_theme.SetLabelField(dist_vertices_shp_f)
end
vertices_theme.SetFixedSizeText(false)
vertTextSym = TextSymbol.Make
vertTextSym.SetFont(Font.Make("Times New Roman","Kursiv"))
vertTextSym.SetSize(7)
vertices_theme.SetLabelTextSym(vertTextSym)
vertLabeler=Labeler.Make(av.GetActiveDoc.GetDisplay.ReturnVisExtent)
vertLabeler.SetLinePlacement(2)
vertLabeler.SetFeatureWeight(#LABEL_WEIGHT_NO)
vertLabeler.SetLabelWeight(#LABEL_WEIGHT_HIGH)
vertLabeler.Load(vertices_theme)
av.GetActiveDoc.GetAutoLabels(vertLabeler,false)
nodes_theme.SetActive(true)
```

Für den Menüpunkt **`Solve`** , Option ***`Start Cumputation`***

script „`locate`“: Dialogfeld für Ausgabedatei und Aufruf des C++ Programmes

```
'************ locate ********
if(_reg_mod_fn=nil) then
   MsgBox.Warning("No actual Parameter File specified","WARNING")
   exit
end
mod_fi=TextFile.Make(_reg_mod_fn,#FILE_PERM_READ)
if(mod_fi=nil) then
   MsgBox.Warning("Cannot open file: "++_reg_mod_fn.AsString,"ERROR")
   exit
end
reg_mod_str= mod_fi.Read(mod_fi.GetSize)
MsgBox.Report(reg_mod_str,"Actual Parameters")
actual=MsgBox.YesNo("Do you want to use these Parameters","Check
Actuality",true)
IF(not(actual)) then
   exit
end
if(_reg_tab_fn=nil) then
   _reg_tab_fn ="reg.txt".AsFileName
end
 _reg_tab_fn=FileDialog.Put(_reg_tab_fn,"*.txt","Regions File")
  if(_reg_tab_fn=nil) then
   MsgBox.Warning("No File for Regions defined","Stop application")
   exit
end
av.ShowStopButton
av.SetWorkingStatus
'actual=MsgBox.YesNo("c:\loc_alloc\locate.exe"++_nodes++_nodes_neighbor++
_reg_mod_fn.GetBaseName++_reg_tab_fn.GetBaseName++_vertices,"Check
Actuality",true)
IF(not(actual)) then
   exit
end
```

```
System.Execute("c:\birdy\c++\locate.exe"++_nodes++_nodes_neighbor++_reg_m
od_fn.GetBaseName++_reg_tab_fn.GetBaseName++_vertices )
MsgBox.Info("Working terminated","Status")
if(_reg_mod_fn <> nil) then
  mod_fi=LineFile.Make(_reg_mod_fn,#FILE_PERM_READ)
  parameter=List.Make
  while (true)
    str=mod_fi.ReadElt
    if(str=nil) then
      break
    else
      parameter.Add(str)
    end
  end
  mod_fi.Close
end
if(VTab.CanMake(_reg_tab_fn)) then
   theVtab=VTab.Make(_reg_tab_fn, false, false)
   mytable=Table.Make(theVtab)
   mytable.SetName(_reg_tab_fn.GetBaseName)
  for each element in parameter
      if ( element.contains("solution=")) then
        sol_str=element.Substitute("solution=","")
       end
  end
   MsgBox.Report(NL+"The solution of the defined System has been saved
to"++
_reg_tab_fn.AsString+NL+NL+sol_str,"Solution found")
    exit
end
MsgBox.Warning("The defined System could not be solved","ERROR")
for each element in parameter
    if ( element.contains("error=")) then
      error_str=element.Substitute("error=","")
    end
  end
MsgBox.Report(error_str,"Error Message")
```

Für den Menüpunkt **`Display Solution`** , Option ***`Select Solution File:`***

script „`sel_reg`“: Dialogfeld für vorhandene Regionsdatei.

```
if(_reg_tab_fn=nil) then
   _reg_tab_fn="*.txt".AsFileName
end
old_reg_tab=_reg_tab_fn

_reg_tab_fn=FileDialog.Show("*.txt","Text","Regions File")
if(_reg_tab_fn=nil) then
  MsgBox.Warning("No File for Regions defined","Stop application")
 _reg_tab_fn=old_reg_tab
  exit
end
mytable=av.GetProject.FindDoc(_reg_tab_fn.GetBaseName)
if(mytable<>nil) then
  exit
end
if(VTab.CanMake(_reg_tab_fn)) then
  theVtab=VTab.Make(_reg_tab_fn, false, false)
  mytable=Table.Make(theVtab)
```

```
  mytable.SetName(_reg_tab_fn.GetBaseName)
  exit
end
MsgBox.Error("Regions file"++_reg_tab_fn++"is not available")
_reg_tab_fn=old_reg_tab
```

Für den Menüpunkt **`Display Solution`** , Option ***`Display Solution`***

script „`show_sol`“: Anzeige der Regionen

```
if((_nodes_shp_fn=nil)or(_reg_shp_fn =nil))  then
  MsgBox.Error("No file names for graphic representation
available","ERROR")
  exit
end
region_doc= av.GetProject.FindDoc(_reg_tab_fn.GetBaseName)
if(region_doc=nil) then
  MsgBox.Error("No tabular information with name "+_reg_tab_fn.
GetBaseName +"available","ERROR")
  exit
end

region_vtab= region_doc.GetVtab
if(region_vtab =nil) then
  MsgBox.Error("Can´t use "+_reg_tab_fn. GetBaseName,"ERROR")
  exit
end

nodes_shp_theme=av.GetActiveDoc.FindTheme(_nodes_shp_fn.GetBaseName)
if(nodes_shp_theme=nil) then
  MsgBox.Error("No graphic representation for centers
available","ERROR")
  exit
end
nodes_shp_ftab= nodes_shp_theme.GetFtab

reg_shp_theme=av.GetActiveDoc.FindTheme(_reg_shp_fn. GetBaseName)
if(reg_shp_theme=nil) then
  MsgBox.Error("No graphic representation for regions
available","ERROR")
  exit
end
reg_shp_ftab= reg_shp_theme.GetFtab
ort_nr_nodes_shp_f=nodes_shp_ftab.FindField("ort_nr")
ort_nr_reg_shp_f=reg_shp_ftab.FindField("ort_nr")
ort_nr_region_tab_f= region_vtab.FindField("ort_nr")
name_nodes_shp_f= nodes_shp_theme.GetLabelField

'**************************removing joins regions ******************
nodes_shp_ftab.UnjoinAll
reg_shp_ftab.UnjoinAll

'*****************************joining regions *********************
nodes_shp_ftab.join(ort_nr_nodes_shp_f, region_vtab, ort_nr_region_tab_f)
reg_shp_ftab.join(ort_nr_reg_shp_f, region_vtab, ort_nr_region_tab_f)
region_nodes_shp_f= nodes_shp_ftab.FindField("region")
reg_shp_theme.SetVisible(true)
reg_shp_theme.SetActive(false)
nodes_shp_theme.SetActive(true)

'******************find center records*********************
```

```
nodes_shp_ftab_bitm= nodes_shp_ftab.GetSelection
nodes_shp_ftab.Query("(([zentrum]=1)and([zentrum_nr]>0))",
nodes_shp_ftab_bitm,#VTAB_SELTYPE_NEW)
nodes_shp_ftab.UpdateSelection

'*******************get center names*******************
ort_nr_l=List.Make
name_l=List.Make
name_regnr_l=List.Make
for each rec in nodes_shp_ftab.GetSelection
  name_regnr_l.Add(nodes_shp_ftab.ReturnValue(region_nodes_shp_f,rec))
  name_l.Add(nodes_shp_ftab.ReturnValue(name_nodes_shp_f,rec))
  ort_nr_l.Add(nodes_shp_ftab.ReturnValue(ort_nr_nodes_shp_f,rec))
end

'**********************legend creating*********************
zentrum_nodes_shp_f=nodes_shp_ftab.FindField("zentrum")
region_reg_shp_f=reg_shp_ftab.FindField("region")

'cent_shp_leg=nodes_shp_theme.GetLegend
reg_shp_leg=reg_shp_theme.GetLegend
reg_shp_leg.load(_reg_leg_fn,#LEGEND_LOADTYPE_ALL)
'cent_shp_leg.load(_cent_leg_fn,#LEGEND_LOADTYPE_ALL)

'********* Put Name of the centers in legend classification ******
reg_shp_leg_class_l=reg_shp_leg.GetClassifications
i=0
for each reg_nr in name_regnr_l
  ort_str="(Nr."+ ort_nr_l.Get(i).AsString+")"
  reg_shp_leg_class_l.Get(reg_nr-
1).SetLabel(name_l.Get(i).AsString+ort_str)
  i=i+1
end

reg_shp_theme.UpdateLegend
av.GetActiveDoc.SetName(_reg_tab_fn.GetBaseName++"regions")
```

Für den Menüpunkt **Display Solution** , Option *Display* Charts for Distances and Demand*:*

script „chart_create“: Darstellung der Zielfunktionswerte und der Nachfragewerte für die einzelnen Angebotsstandorte in Form von Balkendiagrammen für die aktuelle Ergebnisdatei.

```
'******** chart creating ********
'******** getting legend symbols ********
reg_shp_theme=av.GetActiveDoc.FindTheme(_reg_shp_fn.GetBaseName)
reg_shp_leg_sym=reg_shp_theme.GetLegend.GetSymbols

'getting table values
region_vtab= av.GetProject.FindDoc(_reg_tab_fn.GetBaseName).GetVtab
zentrum_region_tab_f= region_vtab.FindField("zentrum")
zent_nr_region_tab_f= region_vtab.FindField("zentrum_nr")
gew_region_tab_f= region_vtab.FindField("demand")
zf_1_region_tab_f= region_vtab.FindField("max_distance")
zf_2_region_tab_f= region_vtab.FindField("demand_over_c1")
zf_3_region_tab_f= region_vtab.FindField("dist/demand_unit")
region_vtab_bitm= region_vtab.GetSelection
region_vtab.Query("([zentrum]=1)", region_vtab_bitm,#VTAB_SELTYPE_NEW) '
and([zentrum_nr]>0))",
region_vtab.UpdateSelection
anz=-1
```

```
for each entry in region_vtab.GetSelection
  anz=anz+1
end
_reg_anz=anz.AsString
gew_field_list={ gew_region_tab_f }', zf_2_region_tab_f }
dist_field_list={ zf_1_region_tab_f }' , zf_3_region_tab_f }

reg_gew_chart=Chart.Make(region_vtab,gew_field_list)
reg_gew_chart.SetSeriesFromRecords(true)
reg_gew_chart.SetRecordLabelField(zent_nr_region_tab_f)

reg_gew_chart_tit=reg_gew_chart.GetTitle
reg_gew_chart_tit.SetName("Demand served by supply location (white
(0)=average)")
reg_gew_chart_tit.SetLocation(#CHARTDISPLAY_LOC_TOP)

reg_dist_chart=Chart.Make(region_vtab,dist_field_list)
reg_dist_chart.SetSeriesFromRecords(true)
reg_dist_chart.SetRecordLabelField(zent_nr_region_tab_f)

reg_dist_chart_tit=reg_dist_chart.GetTitle
reg_dist_chart_tit.SetName("Maximum Distance from supply location (white
(0)=maximum)")
reg_dist_chart_tit.SetLocation(#CHARTDISPLAY_LOC_TOP)

'setting colors from legend colors
reg_gew_chart.GetChartDisplay.SetSeriesColor(0,Color.GetWhite)
 reg_dist_chart.GetChartDisplay.SetSeriesColor(0, Color.GetWhite)
for  each i in 1.._reg_anz.AsNumber 'reg_shp_leg_sym
  reg_gew_chart.GetChartDisplay.SetSeriesColor(i,reg_shp_leg_sym.Get(i-
1).GetColor)
  reg_dist_chart.GetChartDisplay.SetSeriesColor(i,reg_shp_leg_sym.Get(i-
1).GetColor)
end
reg_dist_chart.SetName(_reg_tab_fn.GetBaseName+" distances")
reg_gew_chart.SetName(_reg_tab_fn.GetBaseName+" demand")
reg_gew_chart.GetWin.Open
reg_dist_chart.GetWin.Open
av.TileWindows
```

Für den Menüpunkt **`Display Solution`** , Option *`Clear Solution`:*

script „`clear_reg`“: Löscht die Darstellung der Angebotsstandorte mit ihren Zuordnungen und die Balkendiagramme aus dem Arbeitsspeicher Anzuwenden, bevor weitere Berechnungen durchgeführt werden.

```
 chart_dist= av.GetProject.FindDoc (_reg_tab_fn.GetBaseName+" distances")
chart_weight= av.GetProject.FindDoc (_reg_tab_fn.GetBaseName+" demand")
if(chart_dist<>nil) then
  av.GetProject.RemoveDoc (chart_dist)
  av.GetProject.RemoveDoc (chart_weight)
end
nodes_shp_theme=av.GetActiveDoc.FindTheme(_nodes_shp_fn.GetBaseName)
if(nodes_shp_theme=nil) then
  MsgBox.Error("No graphic representation for centers
available","ERROR")
  exit
end
nodes_shp_ftab= nodes_shp_theme.GetFtab
```

```
reg_shp_theme=av.GetActiveDoc.FindTheme(_reg_shp_fn. GetBaseName)
if(reg_shp_theme=nil) then
   MsgBox.Error("No graphic representation for regions
available","ERROR")
   exit
end
reg_shp_ftab= reg_shp_theme.GetFtab


'*****************************unjoining regions *********************
nodes_shp_ftab.UnjoinAll
reg_shp_ftab.UnjoinAll


'*****************************set regions invisible ********************
reg_shp_theme.SetVisible(false)


'*****************************set centers unselected *******************
nodes_shp_ftab.GetSelection.ClearAll
nodes_shp_ftab.UpdateSelection


'**************** Clear table ***************
tab_reg= av.GetProject.FindDoc(_reg_tab_fn.GetBaseName)
if(tab_reg<>nil) then
   av.GetProject.RemoveDoc(tab_reg)
end
av.GetActiveDoc.SetName("locations")
av.GetActiveDoc.Invalidate
```

Die **Werkzeugleiste** dient einerseits dazu Daten abzufragen, über die graphische Darstellung des Graphen. Andererseits ist es möglich, über die graphische Darstellung Randbedingungen zu spezifizieren. Für diese Funktionen wurden ebenfalls scripts geschrieben.

Werkzeugmenü:

script „`set_fix`“: Spezifikation der Menge der vorgegebene Angebotsstandorte (*specify set of fixed supply locations*)

```
_loc_kind=1
```

script „`set_excl`“: Spezifikation der Menge der ausgeschlossene Angebotsstandorte (*specify set of excluded supply locations*)

```
_loc_kind=2
```

 **script „set_start“:** Spezifikation der Menge der Nachfragestandorte der Startkonfikuration (*specify set of excluded supply locations*)

```
_loc_kind=3
```

 **script „show_loc“:** Anzeige der Nachfragestandorte der aktuellen Menge (s*how specification*).

```
if(_nodes_shp_fn=nil) then
  MsgBox.Error("No file names for graphic representation
available","ERROR")
  exit
end
nodes_shp_theme=av.GetActiveDoc.FindTheme(_nodes_shp_fn. GetBaseName)
if(nodes_shp_theme=nil) then
  MsgBox.Error("No graphic representation for centers
available","ERROR")
  exit
end
nodes_shp_ftab= nodes_shp_theme.GetFtab
ort_nr_f= nodes_shp_ftab.FindField("ort_nr")

nodes_shp_ftab.GetSelection.ClearAll
count=0
count=count+1
if(_loc_kind=1) then
  locations=_vorgegeben
end
if(_loc_kind=2) then
  locations=_ausgeschlossen
end
if(_loc_kind=3) then
  locations=_anfang
end
loc_l=locations.AsTokens(",")
for each entry in loc_l
  abitmap= nodes_shp_ftab.GetSelection
  rec= nodes_shp_ftab.Query("( [ort_nr] = "+entry+"
)",abitmap,#VTAB_SELTYPE_OR)
  'MSgBox.Info(rec.AsString,"rec"+entry)
end
nodes_shp_ftab.SetSelection(abitmap)
nodes_shp_ftab.UpdateSelection
```

 **script „sel_loc“:** Markierte Nachfragestandorte werden zu Standorten der aktuellen Menge (*enter selected locations as set of specified)*

```
if(_nodes_shp_fn=nil) then
  MsgBox.Error("No file names for graphic representation
available","ERROR")
  exit
end
nodes_shp_theme=av.GetActiveDoc.FindTheme(_nodes_shp_fn.GetBaseName)
```

```
if(nodes_shp_theme=nil) then
   MsgBox.Error("No graphic representation for centers
available","ERROR")
   exit
end
nodes_shp_ftab= nodes_shp_theme.GetFtab
ort_nr_f= nodes_shp_ftab.FindField("ort_nr")
if(_loc_kind=1) then
   locations=_vorgegeben
end
if(_loc_kind=2) then
   locations=_ausgeschlossen
end
if(_loc_kind=3) then
   locations=_anfang
end
first=0
'MsgBox.Info(locations,"locations")
if(locations="-1") then
  first=1
end
for each rec in nodes_shp_ftab.GetSelection
  loc= nodes_shp_ftab.ReturnValue(ort_nr_f,rec).AsString
  'MsgBox.Info(loc,"loc")

  if(first=1) then
    locations =loc
    first=0
  else
    locations =locations +","+loc
  end
end
if(locations="") then
  locations ="-1"
end
if(_loc_kind=1) then
   _vorgegeben=locations
end
if(_loc_kind=2) then
   _ausgeschlossen= locations
end
if(_loc_kind=3) then
   _anfang=locations
end
```

script „`clear_loc`“: Löschen aller Standorte der aktuellen Menge (*clear set of specified locations*).

```
if(_nodes_shp_fn=nil) then
   MsgBox.Error("No file names for graphic representation
available","ERROR")
   exit
end
nodes_shp_theme=av.GetActiveDoc.FindTheme(_nodes_shp_fn. GetBaseName)
if(nodes_shp_theme=nil) then
   MsgBox.Error("No graphic representation for centers
available","ERROR")
   exit
end
nodes_shp_ftab= nodes_shp_theme.GetFtab
```

```
nodes_shp_ftab.GetSelection.ClearAll
if(_loc_kind=1) then
   _vorgegeben="-1"
end
if(_loc_kind=2) then
   ausgeschlossen="-1"
end
if(_loc_kind=3) then
   _anfang="-1"
end
nodes_shp_ftab.UpdateSelection
```